LECTURE 3 – VARIABLES

Contents

2.0 Learning Outcomes.

At the end of this topic, the student should be able to:-

1. Define what a variable is, declare and assign it.
2. Read values from the keyboard and display the information on the screen
3. Perform simple mathematic calculations read from the keyboard.

2.1 Introduction

A variable is used by the program to store a calculated or entered value. The program might need the value later, and must then be stored in the computer's working memory.

Example:

Variable name	Variable value
dTaxpercent	0.25

Here we have selected the name 'dTaxpercent' to hold the value 0.25. You can in principle use any variable name, but it is recommended to use a name that corresponds to the use of the variable. When the variable name appears in a calculation the value will automatically be used,

For example:

1500 * dTaxpercent

means that 1500 will be multiplied by 0.25

2.2 Declaring Variables

The purpose of declaring a variable is to tell the program to allocate space in the working memory for the variable. The declaration:

Int iNo;

tells that we have a variable with the name iNo and that is of integer type (int). You must always specify the data type to allocate the correct memory space. An integer might for instance require 4 bytes while a decimal value might require 16 bytes. How many bytes to allocate depends on the operating system. Different operating systems use different amounts of bytes for the different data types.

The variable name should tell something about the usage of the variable. Also, a standard used by many people is to allocate 1-3 leading characters to denote the data type (i for integer).

Note that each program statement is ended by a semicolon.

Below we declare a variable of decimal type:

double dUnitPrice;

double means decimal number with double precision. Compare to float which has single precision. Since double requires twice as many bytes, a double variable can of course store many more decimals, which might be wise in technical calculations which require high precision.

The most common data types

short	integer	Usually 2 bytes
int	integer	Usually 4 bytes
float	decimal	Usually 4 bytes
double	decimal	Usually 8 bytes
bool	true or false	Usually 1 byte

You can declare several variables of the same type in one single statement:

double dUnitPrice, dTotal, dToBePaid;

The variables are separated by commas.

Note that C++ is case sensitive, i.e. a 'B' and 'b' are considered different characters. Therefore, the variables:

dTobepaid

dToBePaid

are two different variables.

2.3 Assignment

Now we have explained how to declare variables, but how do the variables get their values? Look at the following code:

dTaxpercent = 0.25;

iNo = 5;

dUnitprice = 12;

Here the variable dTaxpercent gets the value 0.25, the variable iNo the value 5 and the variable dUnitprice the value 12.

The equal character (=) is used as an assignment operator. Suppose that the next statement is:

*dTotal = iNo * dUnitprice;*

In this statement the variable iNo represents the value 5 and dUnitprice the value 12. The right part is first calculated as 5 * 12 = 60.

This value is then assigned to the variable *dTotal*. Note that it is not the question of an equality in normal math like in the equation x = 60, where x has the value 60. In programming the equal sign means that something happens, namely that the right part is first calculated, and then the variable to the left is assigned that value.

C++ performs the math operations in correct order. In the statement:

dToBePaid = dTotal + dTotal * dTaxpercent;

the multiplication dTotal * dTaxpercent will first be performed, which makes 60 * 0.25 = 15. The value 15 will then be added to dTotal which makes 60 + 15 = 75. The value 75 will finally be assigned to the variable dToBePaid.

If C++ would perform the operations in the stated order, we would get the erroneous value 60 + 60, which makes 120, multiplied by 0.25, which makes 30.

If you need to perform an addition before a multiplication, you must use parentheses:

*dToBePaid = dTotal * (1 + dTaxpercent);*

Here the parenthesis is calculated first, which gives 1.25. This is then multiplied by 60, which gives the correct value 75.

Priority rules:

( )
* /
+ -

2.4 Initiating Variables

It is possible to initiate a variable, i.e. give it a start value, directly in the declaration:

double dTaxpercent = 0.25;

Here we declare the variable dTaxpercent and simultaneously specify it to get the value 0.25.

You can mix initations and pure declarations in the same program statement:

double dTaxpercent = 0.25, dTotal, dToBePaid;

In addition to assigning the dTaxpercent a value, we have also declared the variables dTotal and dToBePaid, which not yet have any values. In the statement:

int iNo = 5, iNox = 1, iNoy = 8, iSum;

we have initiated several variables and declared the variable iSum

2.5 Constants

Sometimes a programmer wants to ensure that a variable does not change its value in the program. A variable can of course not change its value if you don't write code that changes its value. But when there are several programmers in the same project, or if a program is to be maintained by another programmer, it might be safe to declare a variable as a constant.

Example:

const double dTaxpercent = 0.25;

The key word const ensures that the constant dTaxpercent does not change its value. Therefore, a statement like this is forbidden:

dTaxpercent = 0.26;

A constant must be initiated directly by the declaration, i.e. be given a value in the declaration statement. Consequently the following declaration is also forbidden:

const double dTaxpercent;

Read more about Assignment of Values

2.6 The main function

So far we have described code details needed to be able to construct a program. Now we will step back a little and look at the entire program. Look at the following program skeleton

```
void main()
{
    …
    … Various program statements
    …
}
```

To be able to run (execute) a program, a function called **main()**must exist.

A function is a section of code that performs a specific task. Usually a program consists of several functions, but one of them must have the name main(), and the very execution is started in main(). In our first programs we only use one function in each program, and consequently it must be named main(). The parenthesis after main indicates that it is a function. Each function has a parenthesis after the function name, sometimes it is empty and sometimes it contains values or parameters.

A function is supposed to return a value, which could be the result of calculations or a signal that the function turned out successfully or failed. The return value can then be used by the program section calling the function. In our environment it is the operating system Windows that starts the function main(). Windows does not need any return value from our programs, and as a consequence we use the key word void in front of main(). void means that the function will not return any value.

We will discuss functions in a later chapter and will then go deeper into these details. So for now you don’t need to bother about all details, only that you write void main() in the beginning of your programs.

All code belonging to a function must be enclosed by curly brackets, one left curly bracket ({) as the first character and one right (}) as the last. It is also good programming conventions to indent the code between the curly brackets as shown by the example above. The editor in Visual C++ will assist you with the indentation. When you have typed a left curly bracket and pressed Enter, the next and subsequent lines will be automatically indented.

2.7 Input and Output

A program mostly needs data to be entered from the keyboard and results to be displayed on the screen. We will write a little program which displays a request for a value from the user and then reads this value

```
#include <iostream>
using namespace std;
void main()
{
    int iNo;
    cout << "Specify quantity: ";
    cin >> iNo;
}
```

We will soon talk about the first #include statement.

The first thing to be done in this program is that the integer variable iNo is declared. We will need it later in the program.

Then, the text

Specify quantity:

will be displayed on the screen. cout is an abbreviation of console out. With console we mean the keyboard and screen together. The text to be displayed on the screen (Specify quantity) must be surrounded by quote marks (“ “).

The characters << are called stream operator and indicates that each character is streamed to the console. We will return to streams when we work with files in a later chapter It can be hard to remember in which direction to write the stream operator. You can regard it as an arrow directed towards the console, i.e. the characters are streamed out to the console.

The next program statement (cin statement) implies that the program wants something from the console (cin = console in). The program stops and waits for the user to enter a value and press Enter. The value is stored in the variable iNo.

The stream operator >> is here in the opposite direction and indicates that the entered value is streamed the other way, i.e. in to the program.

You may have noticed that the text ”Specify quantity: ” in the cout statement contains an extra space after the colon. This implies that also the space character is streamed to the screen. The visual effect of this is that, when the user enters the number 24, it is not shown close to the text “Specify quantity: “ but appears at a distance of one space character. Write the program and run it, and experiment with space characters and different texts.

#include

cout and cin are functions not automatically available. Therefore we must tell the compiler that these functions are defined in an external file called iostream. When you compile the program, the compiler does not understand the words cout and cin. It will then read iostream and insert the code for how cout and cin should be executed. This is the reason why you the statement

#include <iostream>

must be present first in your program.

Some external files use the extension .h, which is an abbreviation of header file. Another name of such a file is include file. We will use other header files later on when we need particular functions.

namespace

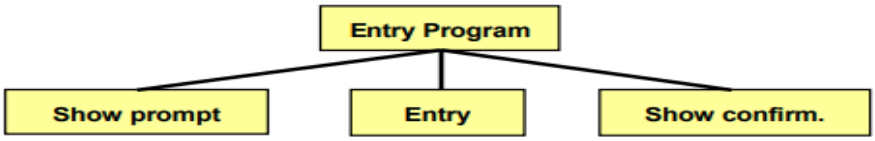
Some development tools, like for instance modern versions of Visual C++, store their include files and classes with code in namespaces. In object oriented programming (which is outside of this course) it is possible to store your own classes in different namespaces. To notify Visual C++ about the include files to be used, reside in the standard namespace, we use the statement

using namespace std;

2.8 An Entry Program

We will now write a program that prompts the user for a number and then shows a confirmation on the screen about which number the user wrote.

To practice algorithm creation, we will first write a JSP graph that shows the program steps before writing the code:



The upper box shows the name of the program (Entry Program). The program contains three steps corresponding to the three boxes which are read from left to right. First we will show the user prompt. The user will then have the opportunity to enter a number. Last, we will show a confirmation about which number the user entered. The program will look like this:

```
#include <iostream>
using namespace std;
void main()
{
   int iNo;
   cout << "Specify quantity: ";
   cin >> iNo;
   cout << "You entered: ";
   cout << iNo;
}
```

First we must include iostream since we are going to read from and write to the console, and indicate the standard namespace to be used. In the main() function we declare the integer variable iNo and then show the text “Specify quantity:

“. The cin statement implies that the program halts and waits for keyboard entry. When the user has entered a number and pressed Enter, the number is stored in the variable iNo.

Then the program continues with displaying the text “You

entered: “ followed by the value of the variable iNo. If the user for instance entered the value 24, the printed text will be

“You entered: 24”. Write the program and run it. Experiment with different entries and other texts.

Note that, even if the program uses two cout statements for the printed confirmation, the result still is one printed line on the screen.

We will now expand the program so that the user can enter a quantity and a price

```
#include <iostream.h>
void main()
{
   int iNo;
   double dPrice;
   cout << "Specify quantity: ";
   cin >> iNo;
   cout << "Specify unit price: ";
   cin >> dPrice;
   cout << "You entered the quantity " << iNo <<
            " and the price " << dPrice;
}
```

We have used an integer variable for the quantity and a double variable for the unit price, since the price could require decimals.

Note that, when entering a decimal value, you must use a decimal point, not decimal comma.

The last cout statement contains some news; you can combine several texts and variable values into one single statement, provided that you use the stream operator between every text and variable. We have split the statement on two lines, but you can write the whole statement on one single line. Blanks or line breaks in the code has no effect on the displayed result.

You could also combine the entry of quantity and price in the following way:

```
cout << "Specify quantity and unit price: ";

cin >> iNo >> dPrice;
```

First the text prompt is displayed to the user. When the program halts and waits for entry, you can enter a quantity and unit price with a space character in between, or press Enter. The first entered value is stored in the variable iNo and the second in dPrice.

If you press Enter after the first entered number, the program will still be waiting for yet another value. You must also

enter the unit price before the program can continue the execution.

If you want a line break in the displayed result, you can use the keyword endl:

```
cout << "You entered the quantity " << iNo <<

 endl << "and the price " << dPrice;
```

The statement implies that the printed result will look like this:

You entered the quantity 5 and the price 12.45

2.9 Formatted Output

When programming for a DOS window, which we have done so far and will do during the rest of the course, there are very limited possibilities for a nice layout of printed information, especially if you compare to common Windows environment.

C++ however offers a few functions for improved control of printed information. We will discuss the following:

• **Fixed / floating decimal point**. When printing large numbers, it might happen that C++ uses floating

decimal point. For instance the number 9 560 000 000 (fixed) might be printed as 9.56E+9, which

is interpreted as 9.56 x 109 (floating decimal point). C++ itself controls when to use either of these

representations. Many times we want to control this ourselves.

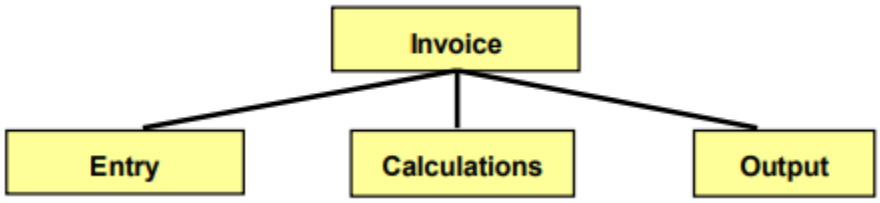
2.10 Invoice Program

We will now write a program where the user is prompted for quantity and unit price of a product, and the program should respond with an invoice receipt like this:

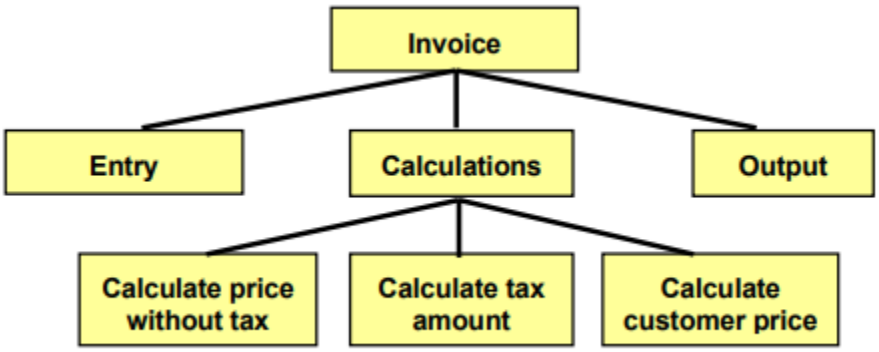
```
INVOICE

=======

Quantity:          30

Price per unit:    42.50

Total price:       1593.75

Tax:       318.75
```

The program should thus calculate the total price and tax amount.

We start with a JSP graph:



First, the user will enter quantity and unit price of the product (the Entry box). Then we will calculate the total price and tax amount (the Calculations box). Last, the information will be printed (the Output box). The first and last boxes are pretty uncomplicated, but the Calculations box requires that we go deeper before starting to code. Input data is quantity and unit price. We multiply these, which gives the price without tax. We then multiply this amount by the tax percent, which gives the tax amount. Finally we add these amounts to get the customer price. The detailed JSP graph will then look:



```
#include <iostream>
#include <iomanip>
void main()
{
 //Declarations
 int iNo;
 double dUnitPr, dPriceExTax, dCustPrice, dTax;
 const double dTaxPerc = 25.0;
 //Entry of quantity and unit price
 cout<< "Specify quantity and unit price: ";
 cin >> iNo >> dUnitPr;
 //Calculations. First the price without tax
 dPriceExTax = dUnitPr * iNo;
 //then the tax amount
dTax = dPriceExTax * dTaxPerc / 100;
 //and finally the customer price
 dCustPrice = dPriceExTax + dTax;
 //Output
 cout << endl << "INVOICE";
 cout << endl << "=======" << endl;
 cout << "Quantity: " << setw(5) << iNo << endl;
 cout << setprecision(2) << setiosflags(ios::fixed);
 cout << "Price per unit:" << setw(8) << dUnitPr << endl;
 cout << "Total price: " << setw(8) << dCustPrice <<endl;
 cout << "Tax: " <<setw(8) << dTax << endl;
}
```

As you can see we have inserted comments in the code. Comments don't affect the final size of the program or performance.

Therefore, use comments frequently, partly to explain the operations to others, and partly as a check list at maintenance of the program some years later.

Comments are surrounded by the characters /* and */. All text between these delimiters is treated as comments. You can have as many lines as you want between these delimiters. Another way is to begin a comment line with //, Then only that line will be treated as comment.

After the include statements the required variables are declared. The variable iNo is an integer, while all variables capable of storing an amount have been declared as double. The tax percent is declared as constant, since it should not be amended in the program.

Compare the code to the JSP graph and you will discover that we have followed the sequence of the boxes when coding

2.11 Summary

In this chapter we have taken our first stumbling steps in C++ programming. We have learnt what a variable is, how it is declared and assigned a value. We have also learnt to read and write data, and in connection to that, also present data in a more user-friendly way by means of formatted output. We have learnt how to include header files and we have written some example programs utilizing specialties like the modulus operator, type casting and random number generation.

But above all we have practiced how to build a solution to a problem by means of algorithms and JSP graphs. And this will be still more important when we enter into the subject of the next chapter, selections and loops.

2.12 Exercises

1. Originate from the Entry program and extend it so that the user can enter two numbers. Both numbers should then be printed on the screen by the program.

2. Write a program that prompts the user for two numbers and prints their sum.

3. Extend the previous excercise so that the program prints the sum, difference, product and quotent of the two numbers.

4. Write a program that prompts the user for 3 decimal numbers and then prints them on the screen with the decimal points right below each other.

5. Originate from the Invoice program and amend it so that:

    a) the user can enter a tax percent.

    b) a 10% discount is deducted before the tax calculation

    c) the last cout statement is organized in a more structured way

    d) The price with tax excluded is printed

e) the discount amount is printed.

6. Write a program that prompts the user for the gas quantity and the gas price per litre. The program should then print a gas receipt like this:

```
 RECEIPT
 ====================
Volume:          45.24 l
Litre price:     9.56 kr/l
---------------------------------
To be paid:      432.49 kr
```

7. Write a program that prompts the user for current and previous electricity meter value in kWh and the price per kWh. The program should then calculate the total price of the current consumption.

8. Write a program that prompts the user for five integers. The program should then print:

a) the sum of the integers

b) average

c) the sum of the squared numbers

d) the sum of the cube of the numbers

9. Write a program that prompts the user for a number, divides it by 3 and prints the result in the form: “4 and remainder 2”.

10. You want a program that converts a temperature in Celsius to Fahrenheit according to the formula:

tempF = 1.8 * tempC + 32

Create the conversation with the user in your own way.