# UNIT 17 INTRODUCTION TO PROGRAMMING

**Structure** **Page No.**



## 17.1 INTRODUCTION

In this Unit, we introduce you to the concept of algorithms and explain the principles of program design in Sec. 17.2. In Sec. 17.3, we will explain how to compile C Programs. We will also explain how to use Dev-C++ and Anjuta for compiling C programs.

### Objectives

After studying this unit, you should be able to

- explain the concept of algorithm;
- make flow charts for simple programs;
- compile C programs from the command line; and
- use Anjuta and Dev-C++ for writing C programs.

## 17.2 DESIGN OF PROGRAMS

You are already familiar with problem solving in Mathematics and you are probably familiar with the steps involved. Even if you do not consciously follow certain prescribed steps, you will carry out the following steps:

1. Understand the problem. This will involve asking the questions like:
    - What are the conditions?
    - What is to be proved or found?
    - Can the problem be solved meaningfully with the conditions given?
2. Try to find some examples of the problem.
3. Try to think of a similar problem that you have solved before.

All these considerations are equally valid in the design of computer programs. The steps in designing a program to solve a particular problem are

1. Understand the problem.
2. Work out the steps involved in solving the problem.
3. Write a program that carries out the steps involved in solving the problem.

4. Check if the program works correctly by trying out the program for those cases that can be verified with manual computation.

The steps 2 and 3 in solving a mathematics problem gets included in step 2 of program design. Also, the steps worked out in step 2 has to be in a specific form. It should be in the form of an *algorithm*.

Loosely speaking, an algorithm is a step by step procedure for accomplishing certain task. The task could be anything, cooking some dish, calculating income tax, etc.

To understand the concept of algorithm, let us now take an example of a task and write an algorithm for it. Suppose we want to factorise a quadratic polynomial $x^2 - ax + b$ with integer coefficients over integers. Let us assume that a and b are positive for simplicity. How do we do it if we do it by hand? We look for two numbers p and q such that $p + q = a$ and $pq = b$. Another way of achieving the same thing is to look at divisors of b. If, for some divisor d of b, $d + \frac{b}{d} = a$, d and $\frac{b}{d}$ are the roots of the equation. So, $(x - d)\left(x - \frac{b}{d}\right) = x^2 - ax + b$ and we are done. Otherwise, we cannot factorise the polynomial over the integers.

Let us now write this as a step by step procedure:

1. $d \leftarrow 1$.(Take $d = 1$.)
2. [Look for divisors of b.] Check whether $d \mid b$. If $d \mid b$, go to Step 3.

   If $d \nmid b$, check if $d + 1 < b$. If $d + 1 < b$, increase the value of d by 1 and carry out Step 2 for this new value of d. If $d + 1 \geq b$, stop; we have already checked all the divisors of b and so we cannot factorise the polynomial over the integers.
3. [Finished ?] If $d \mid b$, check if $d + \frac{b}{d} = a$. If yes, the factors are $(x - d)$ and $\left(x - \frac{b}{d}\right)$. Otherwise $d \leftarrow d + 1$(Increase the value of d by 1.) and go to Step 2.

Let us see how to carry this out for the polynomial $x^2 - 8x + 15$ (See Fig. 1.) So, do we

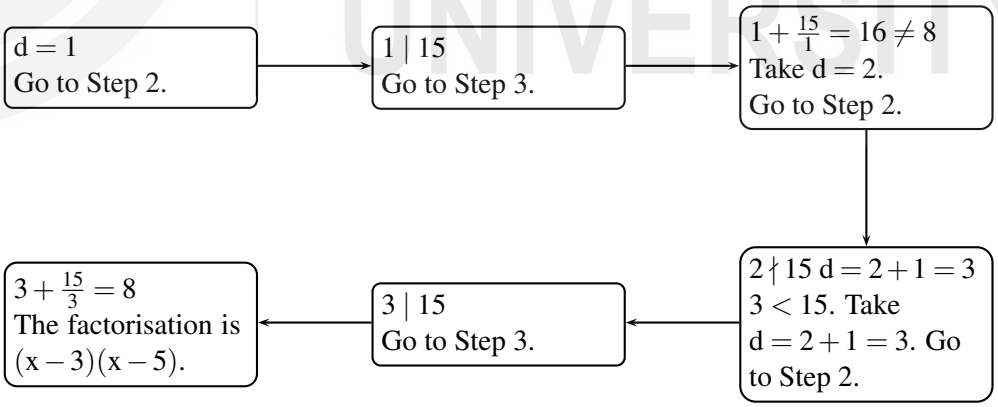


**Fig. 1: Steps involved for $x^2 - 8x + 15$**

have an algorithm? Before we can say that, we have to check whether our procedure has certain characteristics. They are:

**Finiteness** The algorithm has to terminate after a finite number of steps. Our procedure satisfies this because our algorithm will certainly stop after checking all the divisors of b, which are finite in number; it may stop earlier if it finds a divisor which is a root of the polynomial.

**Definiteness** The steps involved must be precisely defined. There should be no ambiguity in the meaning. This is so in our procedure.

**Input** An algorithm has zero or more inputs; the information we give to the algorithm at the start. In our procedure, the coefficients a and b of the polynomial are the input.

**Output** An algorithm has one or more outputs. In our procedure, the output could be the factors or the information that we cannot factorise the polynomial over integers.

**Effectiveness** This means that the operations performed must be simple enough that anybody can perform them using a paper and pencil in a finite amount of time. As you can see, the operations in our procedure can be performed with pencil and paper.

In the case of computer algorithm, effectiveness means that we should be able to carry out the steps in the procedure on a computer.

Try the following exercises to check your understanding.

---

E1) Make a chart of the steps involved in factoring the polynomial $x^2 - 7x + 12$.

---

The steps of an algorithm can also be shown using a **flow chart**. Let us look at the flow chart corresponding to the algorithm for factoring a polynomial (See Fig. 2). Note that,

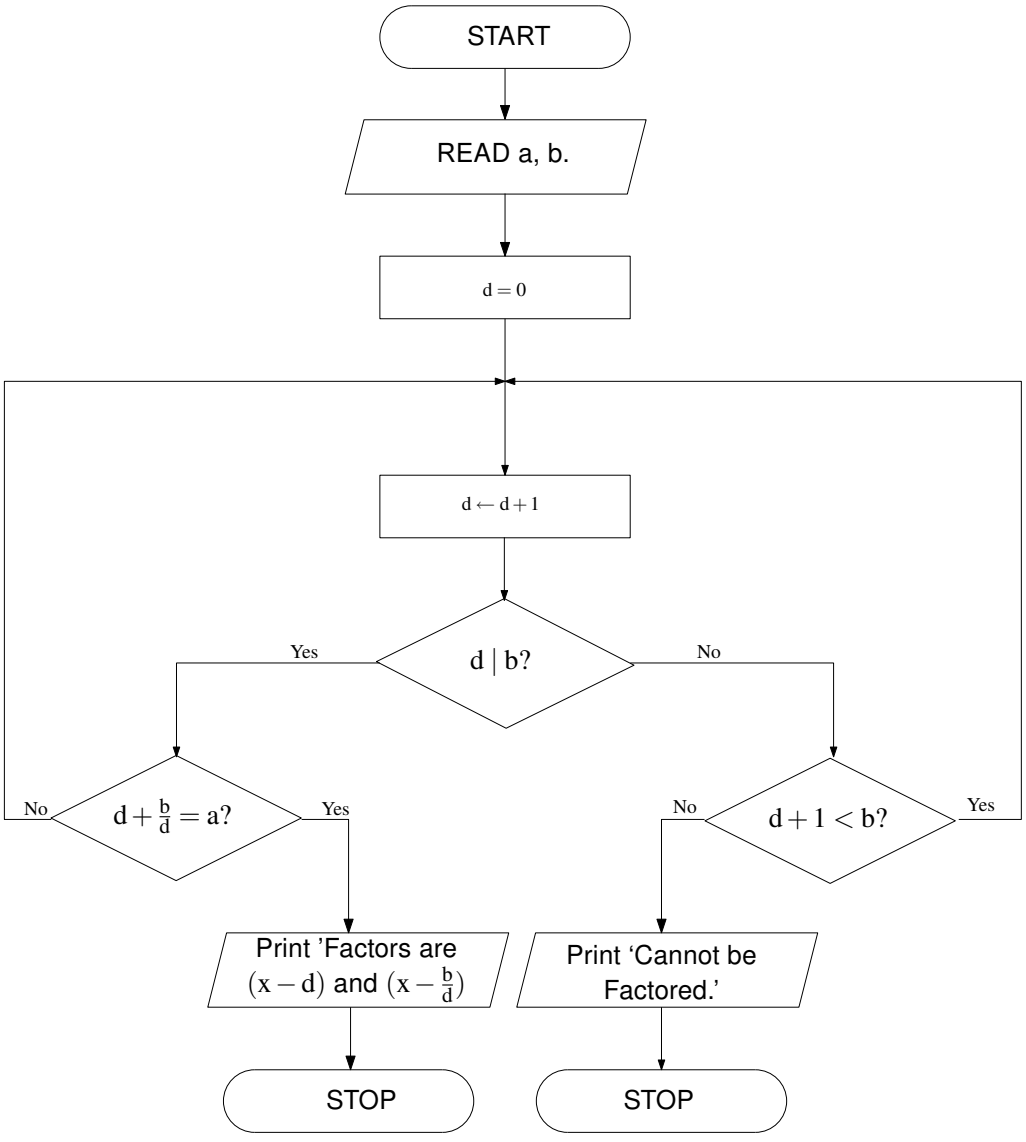


**Fig. 2: Flow chart for the polynomial factoring algorithm.**

in the flow chart, there are different kinds of boxes.

| | |
|---|---|
| **Parallelograms** | These are used for input and output operations. |
| **Rectangles** | These are used to show any processing, arithmetic, or storage operation. |
| **Diamond** | Diamond boxes are used to show questions asked or conditions tested, based on whose answers appropriate exits are taken by the procedure. The exit from the diamond shaped box are labelled with answers to the questions. |
| **Rounded Rectangles** | These are used to show beginning or end point of the procedure. |

Let us now look at one more example. We will discuss a simple algorithm to test whether a given natural number is prime or not. We can do this by checking whether any of the natural numbers less than n divide n. We can actually do a little better. It is enough to check that the number has no proper divisor less than $\sqrt{n}$. Why is this so? If n is composite, $n = pd$, where p is the smallest prime divisor of n. Since $d \neq 1$(because n is not a prime), it has a prime divisor which is bigger than p by the choice of p. So, $n = pd \geq p^2$ or $p \leq \sqrt{n}$. So, if n is composite, it has a divisor not bigger than $\sqrt{n}$. Let us write the steps in the algorithm first.

1. $d \leftarrow 2$
2. [Check whether d is smaller than square root of n.] Check if $d \leq \sqrt{n}$. If 'Yes', go to Step 3. If 'No', n is a prime; stop.
3. [Check if $d \mid n$.] If $d \mid n$, n is not a prime; stop. If $d \nmid n$, $d \leftarrow d + 1$ and go to step 2.

The flowchart for this algorithm is given in Fig. 3. The examples that we have given are simple ones because we wanted to explain the concept of an algorithm. What happens if the procedure is complicated? In this case the procedure is broken into small manageable parts or modules. We write program instructions for the each of the modules, keeping in mind the role these modules have to play in solving the main problem.

Let us look at an example now. Suppose we want to write a program to solve a system of n simultaneous linear equations in n unknowns using Gauss elimination method. In case you have not studied this before, here is a quick introduction. You can look at Unit 5, Block 2 of the course MTE-10 or any other book in numerical analysis for details. This method is a generalisation of the familiar method of eliminating on unknown between a pair of simultaneous equations. In this we use row operations to reduce the system of equations to triangular form. There are three tasks to be carried out in this method. They are

1. Carrying out row operations.
2. After carrying row operation on the the first $i - 1$ rows, check if there is a row in which the ith variable has a non-zero coefficient. If there is no such equation, stop the procedure saying that the system does not have a unique solution.
3. After successfully carrying out the procedure $n - 1$ times check if the coefficient of the nth variable is non-zero. If it is, find the solution to the equations. Otherwise, stop with an error message.

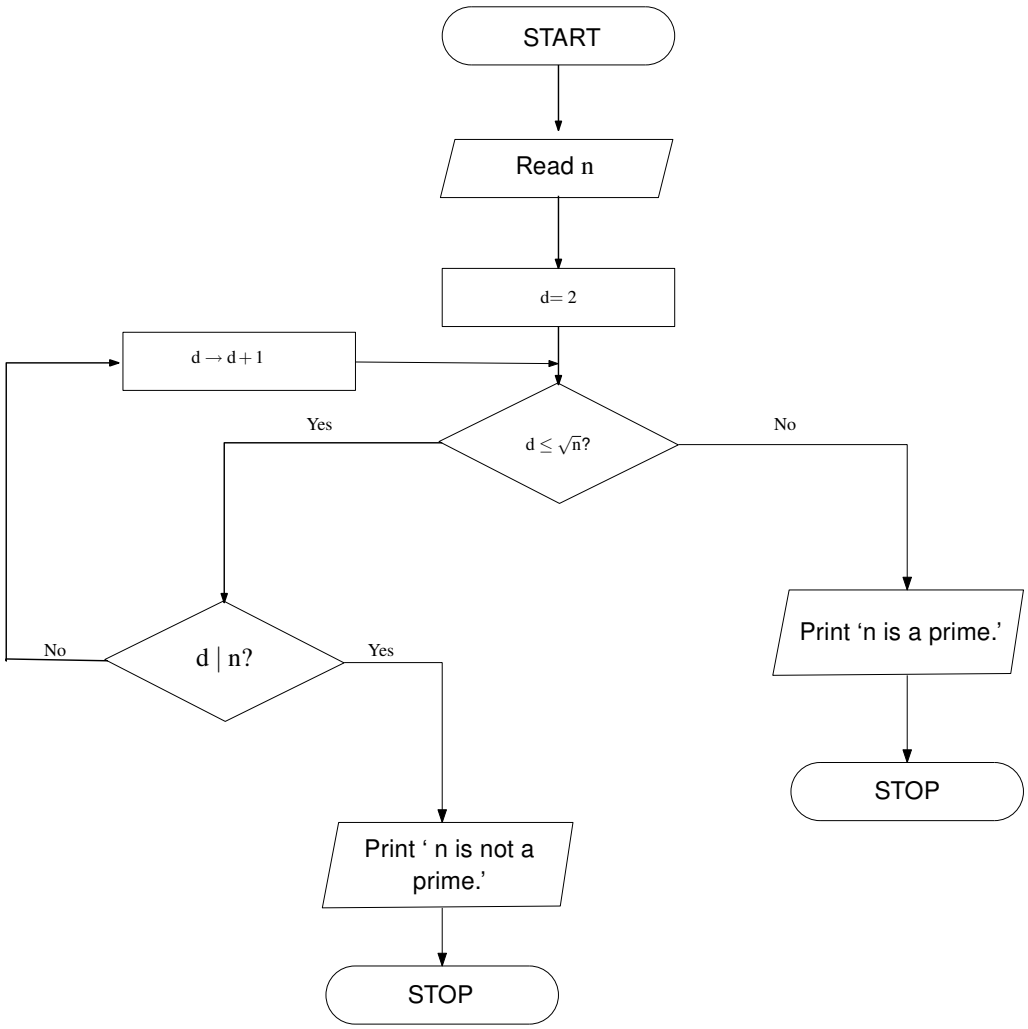


**Fig. 3: Flowchart for primality testing algorithm.**

When we write the program, we write functions to carry out these tasks. Of course for a complete program, we also need a function that takes the input and another one for giving the output.

In the next section, we will see how to compile C programs.

## 17.3 COMPILING C PROGRAMS

In this section, we will see how to use the GCC compiler which is free and available for many operating systems. The simplest way to compile a C program using GCC is from the command line. In windows, you can open a command line terminal by clicking on `Start` ⇒ `run` and typing 'command' in the dialogue box that comes up and pressing the Enter key. See Fig. 4 on the next page.

GCC stands for GNU Compiler Collection.

In most versions of linux you can click the right mouse button on the desktop to get the menu shown in Fig. 5 on the following page. By clicking the left mouse button on the 'Open Terminal' in this menu, you can open a linux command line window, also called a terminal in linux.

Your instructor will help you to open a command line terminal.

To compile the first program unit1-prog1.c you have to type

```
gcc unit1-prog1.c -o unit1-prog1
```

in the command line and press enter. You will have to type the full path of gcc if it is not on your path in the case of windows. The part `-o unit1-prog1` specifies the name of the output file. Without this, in linux, a file called a.out is created.

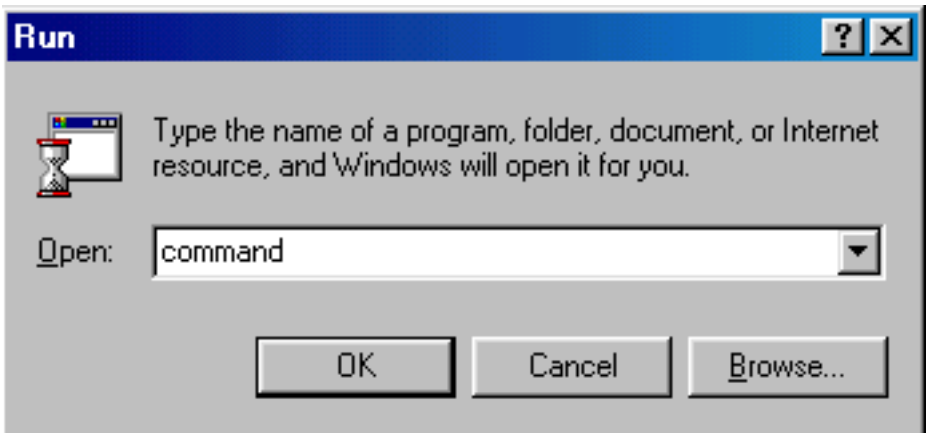


**Fig.4: Opening a command line in windows.**


**Fig. 5: Opening a command line window in linux.**

You can execute this file by typing `./a.out` in the command line. If you compile another program without mentioning the output file name, it will overwrite the file created earlier. If you add the part `-o unit1-prog1`, an executable file with the name unit1-prog1 will be created in the case of linux. You can execute it by typing `./unit1-prog1`. In windows, a file with name `unit1-prog1.exe` will be created. You can run this by typing the file name `unit1-prog1` and pressing the Enter key.

You can add some more options. One of them is `-Wall`. This asks GCC to print all the warnings. In the initial stages, this option will be helpful in finding the mistakes in your program. In case you want to use the maths library you have to add `-lm` to the command line. If you want the GCC compiler to use only features specified in ANSI specification, you can add the option `-ansi` to the command line. Using this option will ensure that your program will compile correctly in any ANSI compliant compiler. For example, if you want to compile the program unit1-quad.c which uses the square root function `sqrt()` and the absolute value function `fabs()`, the command is

```
gcc -Wall -lm -ansi unit1-quad.c -o unit1-quad
```

If you are writing a program, this procedure of compiling from the command line is very tedious. It is more convenient to use an IDE(Integrated Development Environment) for writing programs. IDEs provide a convenient means of writing, compiling and debugging programs. We will discuss two IDEs, one for the linux and one for the windows. For linux, we recommend Anjuta, which provides a very user friendly way of writing and compiling C programs. You can start Anjuta by clicking on

the bottom left corner of the desktop and selecting Anjuta under ‘development’. A window similar to the one in Fig. 6 opens.

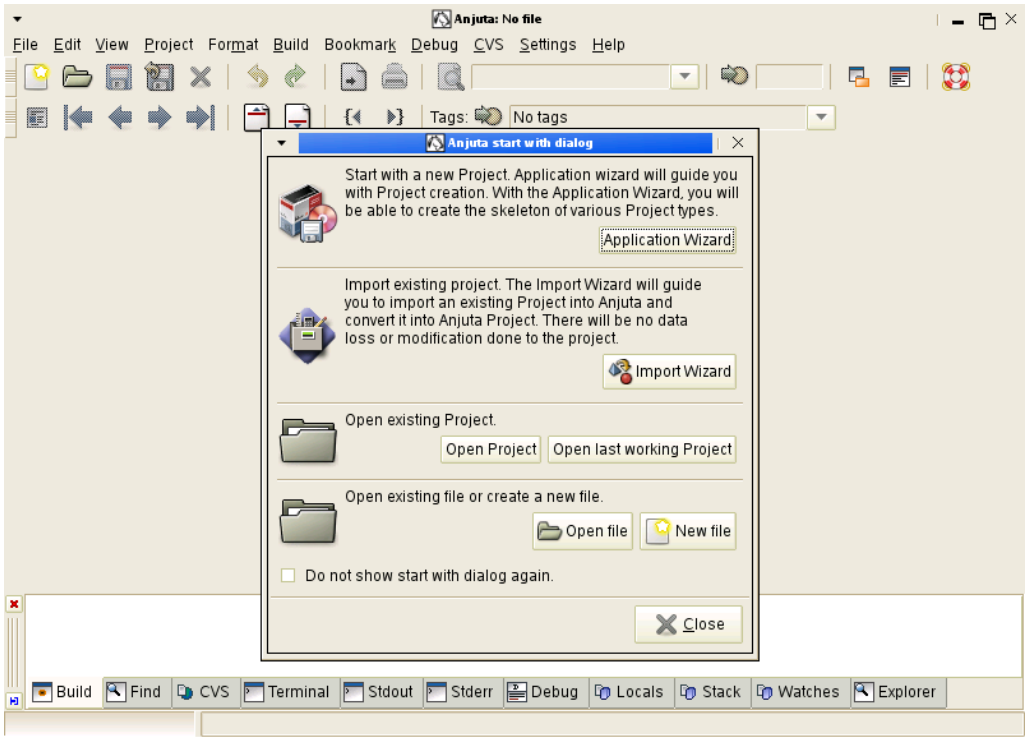


**Fig.6: Anjuta start up window**

If you want to open an already existing file, click on ‘Open file’, go to the directory containing your file and open it. If you want to write a new program, click on ‘New file’. A new file opens for you to type your program.

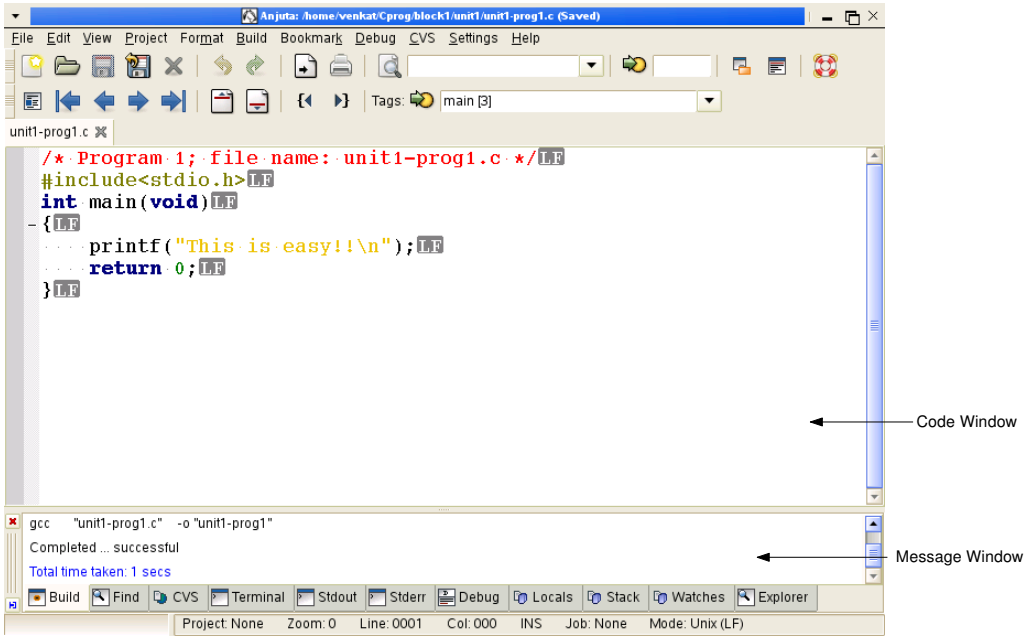


**Fig.7: Anjuta Window with a program.**

After you finish writing the program, press F9 . If the program compiles successfully, press F11 to link. If there are any errors in your program, there will be an error message in the message window. You can double click on the message to go to the line containing the error. After you successfully compile your program, you can then press F3 to run the program. The output is similar to the one in Fig. 8 on the following page.To set options of the compiler options click on `Settings` ⇒ `Compiler and`

`linker settings ...`

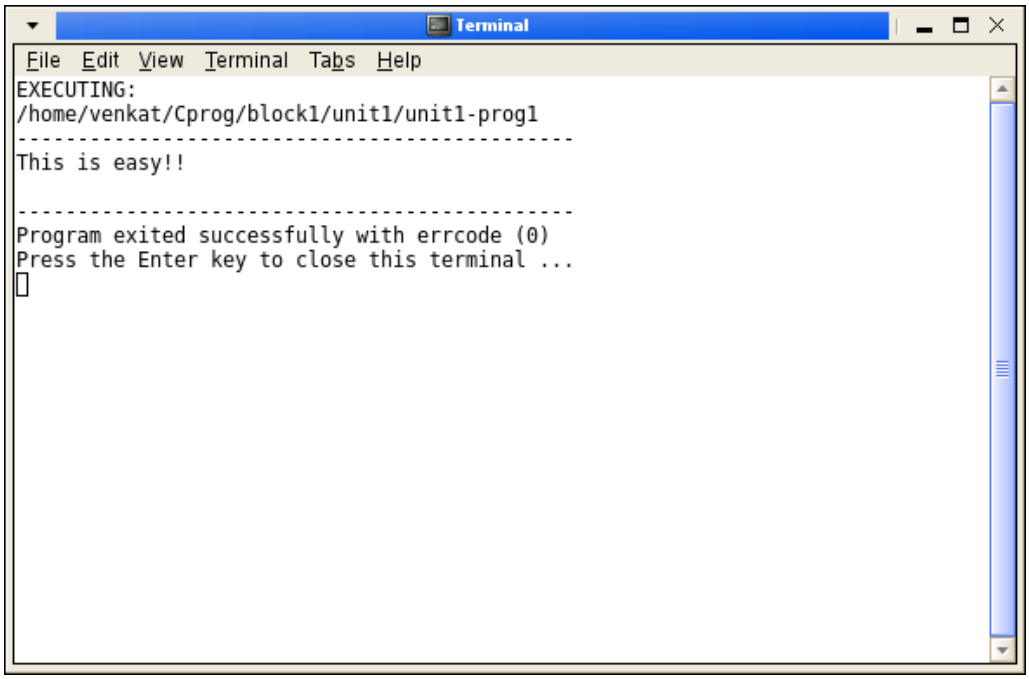


**Fig.8: Output window.**

In the dialogue that opens, click on the 'Warnings' tab and then double click on 'Wall' to enable all warnings. See Fig. 9.

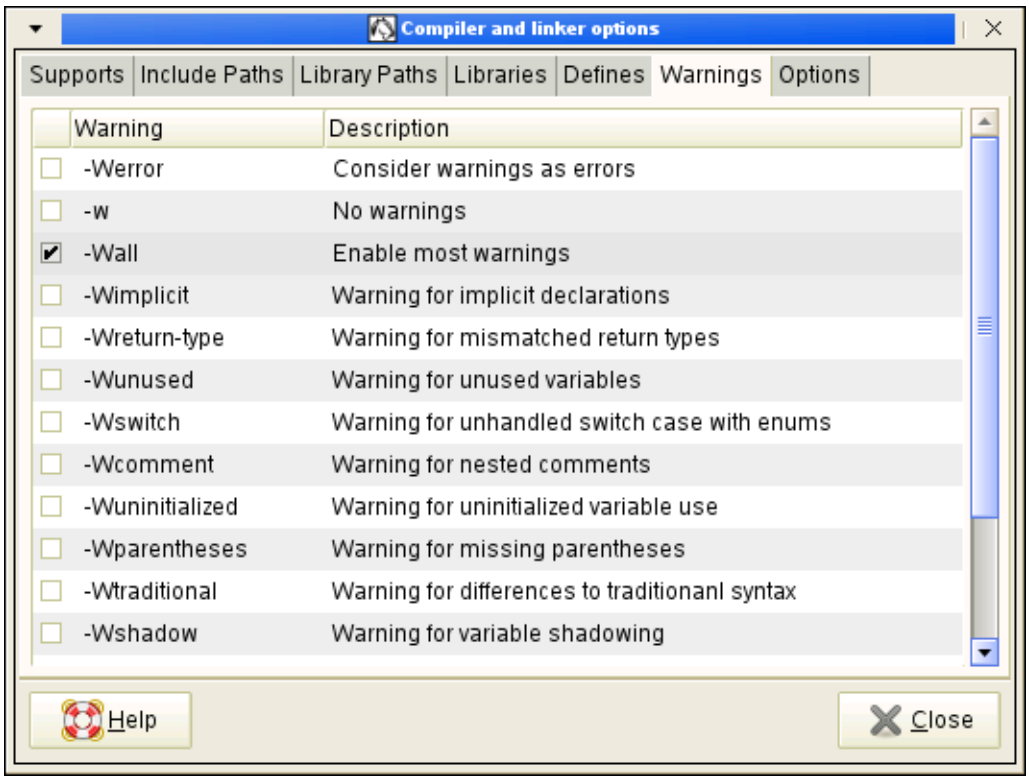


**Fig.9: Setting warnings.**

To link the maths library, click on the 'Libraries' tab and enter 'lm' on the space given. Click '+ Add' afterwards. See Fig. 10 on the facing page.

In windows, you can start Dev-C++ by pressing `START` ⇒ `Programs` → `Blood shed Dev-C++` → `Dev-C++`. The window is similar to Fig. 11 on the next page.To compile and run a file, press F9 . If you just want to compile, but not run, press Ctrl + F9 . To just run press Ctrl + F10 .

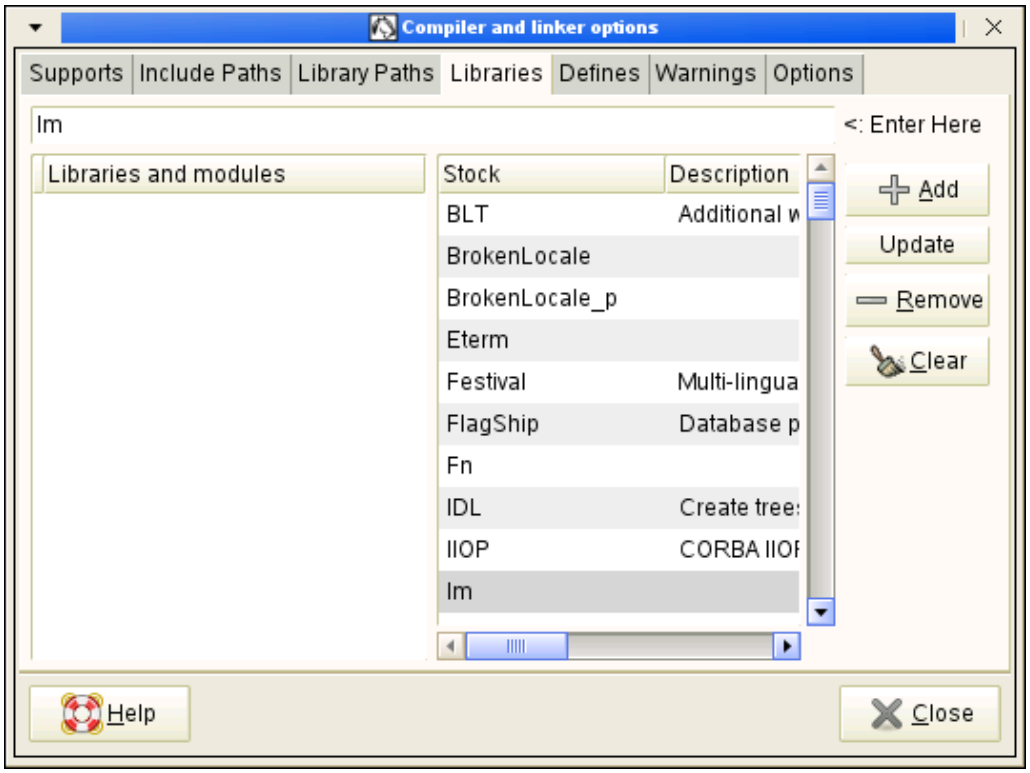


**Fig.10: Linking maths library.**

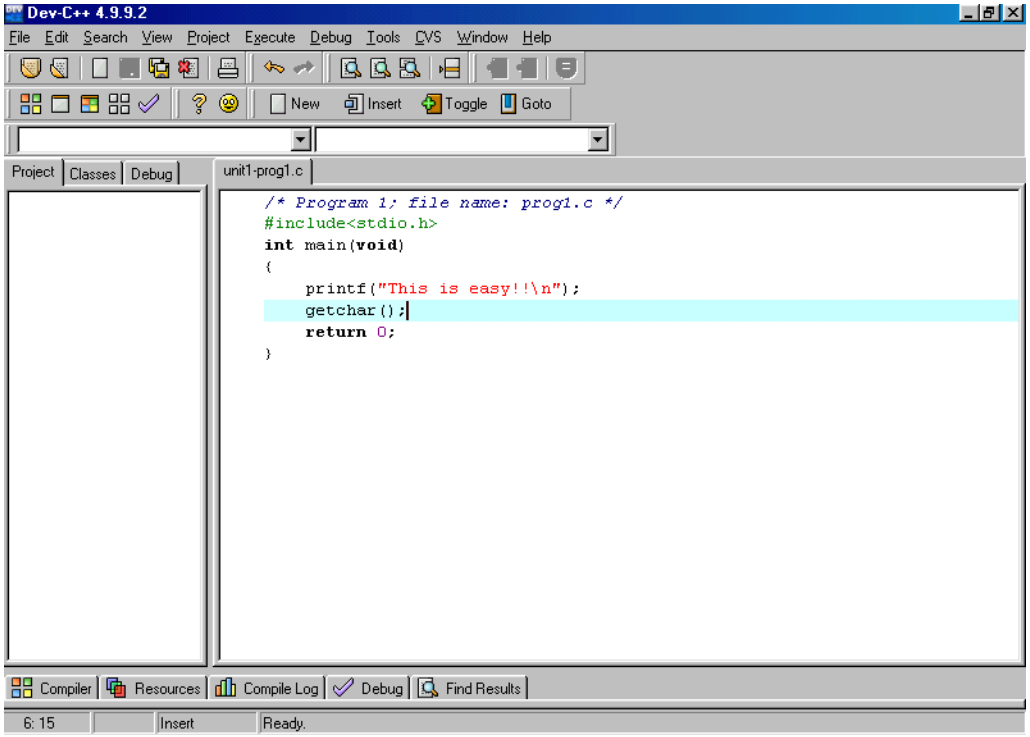


**Fig.11: Dev-C++ startup window.**

To set the command line options, press `Tools` ⇒ `Compiler Options` and enter the options as shown in Fig. 12 on the following page.

Note that, in the case of Dev-C++, you have to add the line `getchar();` before the **return** statement; otherwise the output window will close immediately after running the program and you will not be able to see the output. Also, if your program has a `scanf()` statement, get rid of the trailing <CR> you typed after the input using a `getchar()`. See page 48 of Block 2 for the reasons for doing so.

We will now briefly discuss how to programs distributed over multiple files. Let us see

how to compile the 3 files given in exercise 6 of Block 2. The names of the files are unit8-prog9-f1.c, unit8-prog9-f2.c and unit8-prog9-f3.c.

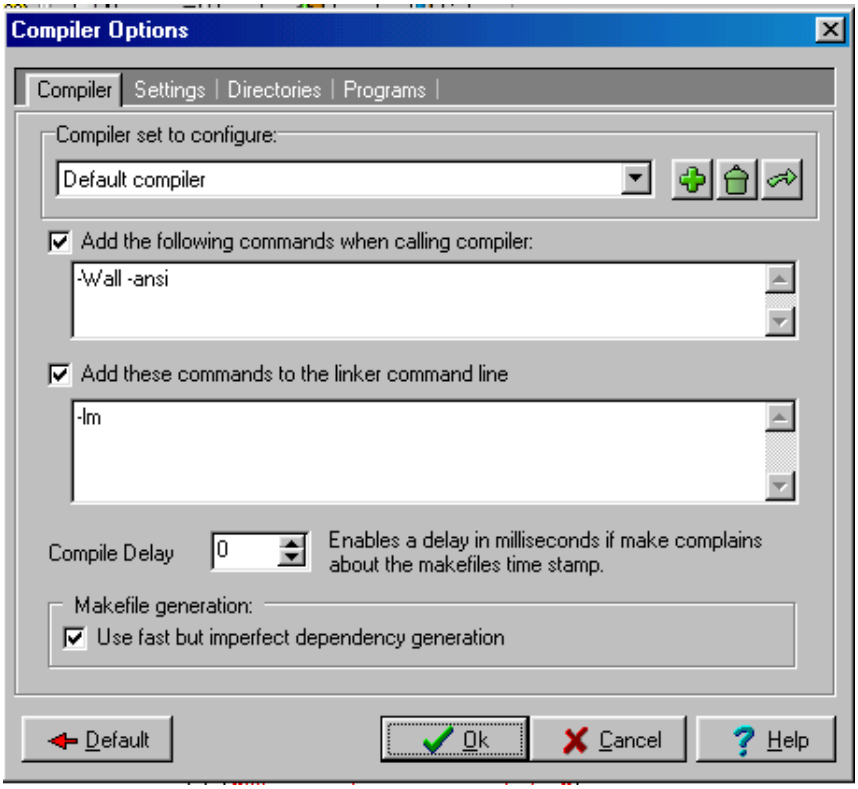


**Fig.12: Compiler options for Dev-C++.**

In this case we first create object files using the `-c` option:

```
gcc -c unit8-prog9-f1.c unit8-prog9-f2.c unit8-prog9-f3.c
```

To create an executable file called unit8-prog9, give the command

```
gcc unit8-prog9-f1.o unit8-prog9-f2.o unit8-prog9-f3.o
   -o unit8-prog9
```

The command is shown in two lines, because the line is too long. You have to type it in one line. Ignore any warnings printed by the compiler. Now, you can run the program by typing `./unit8-prog9`. For more complicated projects, you will have to create make files. See page 532 of the book *A Book on C, fourth edition*, by *A. Kelley and I. Pohl* to learn about make files. This book will be available in your study centre library.

In the next session, we will summarise this unit.

## 17.4 SUMMARY

In this unit we have discussed

1. the concept of algorithm;
2. how to make flow charts for simple programs;
3. how to compile C programs from the command line; and

4. how to use Anjuta and Dev-C++ for writing C programs.

## 17.5 SOLUTIONS/ANSWERS

E1) See Fig. 13

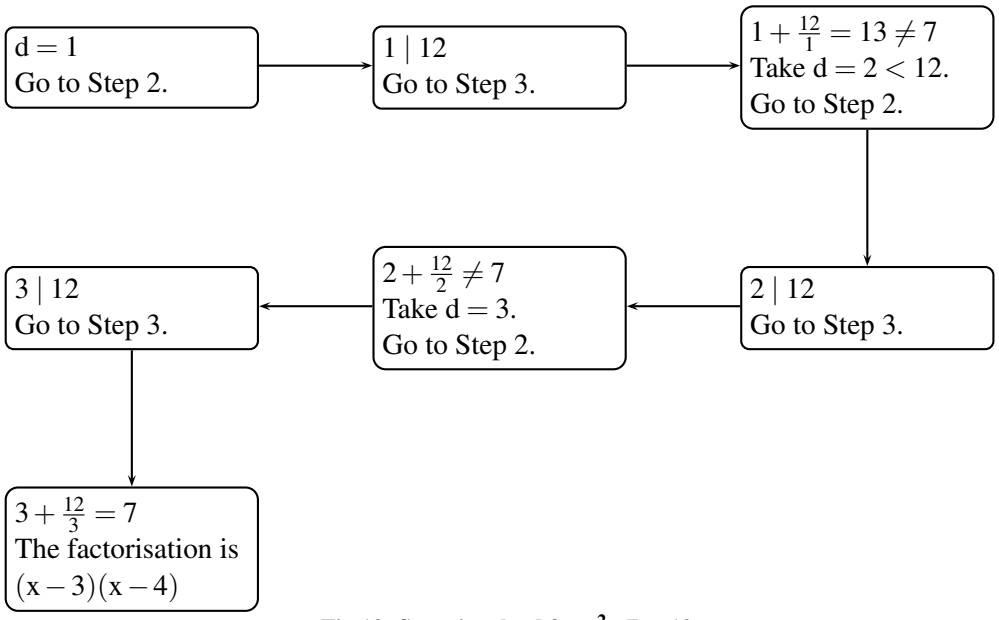


**Fig.13: Steps involved for $x^2-7x+12$**