A. How to Get Started with CoCoA

No keyboard present.
Hit `F1` *to continue.*
(DOS Error Message)

A.1 Getting CoCoA

The computer algebra system CoCoA is available free of charge via the internet. You may download it from the WWW page

`ftp://cocoa.dima.unige.it/cocoa/index.html`

Choose the file corresponding to your machine and operating system, and do not forget to get a copy of the manual from the directory `/doc`. Then decompress the file ending in `.zip`, or `.tgz`, or `.hqx`. Depending on your system, you will have to use a command like `unzip` or `tar -zxvf`, or the graphical interface of a decompression tool like `Winzip`.

As a result you should have an appropriately named directory or folder, for instance `/usr/local/cocoa/`, which contains the executable file of the program (called `cocoa` or `cocoa.exe`), a file named `userinit.coc`, and a subdirectory called `lib`. If, instead, you find hundreds of files in your folder, you forgot to uncompress in such a way that all paths are restored!

A.2 Starting CoCoA

Now let us try the program. You can start it by simply typing

```
cocoa
```

while you are in the folder containing the executable file. Or, if you have a graphical operating system, you can also click on its icon. And if there is no icon, create a link or shortcut to the executable file first. Then the current version of CoCoA starts up and displays a screen like

```
--------------------------------------------------------
---           ___/       ___/         \              ---
--           /      _ \ /      _ \    , \             --
--           \    |   | \    |   |   ___ \            --
---          ____, __/  ____, __/ _/    _\           ---
--------------------------------------------------------
--     Version      : 4.0                             --
--     Online Help  : type  Man();                    --
--     Web site     : http://cocoa.dima.unige.it      --
--------------------------------------------------------

-- The current ring is R ::= Q[x,y,z];
--------------------------------
```

Below we shall explain how you can enter commands by going with you through a sample session. But first, let us show you how you can exit the program. Simply type

```
Quit;
```

hit `<Enter>`, and that's it! Or maybe not. If you are using the graphical interface of CoCoA under Unix or Windows, you can either send a line to CoCoA by typing `<Ctrl + Enter>` at the end of the line, or you can select a line or a group of lines and send it to CoCoA using `<Ctrl + Enter>`. And if you are using CoCoA on Macintosh, you should make sure you hit the `<Enter>` key, not the one marked `<Return>`. And if, by any chance, you are in a mediterranean mood, you can also leave CoCoA with a cordial

```
Ciao;
```

A.3 Using CoCoA Interactively

In fact, when you typed `Quit;` above, you entered your first CoCoA command. A CoCoA command has two notable features: it starts with a capital letter, and it ends with a semicolon. If you want to define a new object, you have to give it a name starting with a capital letter. Then you can assign it a value by using `:=` as in the following example.

```
F:=x^2+y^2-1;
```

This command defines a new polynomial `F` in the current ring which has the value x^2+y^2-1. (We will be talking about rings later, but for the moment we assume that you just started CoCoA and that the current ring is the default ring `R = Q[x,y,z]`.) The value of the new polynomial `F` is `x^2+y^2-1`. As you can see, the indeterminates of a polynomial ring in CoCoA have lower-case letters as names and are raised to powers using the `^` key. Sometimes they carry indices like x[1],x[2],x[3]. All other names of objects and functions in CoCoA start with a capital letter.

Now let us print out the value of our polynomial F. For this we could type Print(F); or simply

```
F;
```

Then CoCoA answers

```
x^2 + y^2 - 1
-------------------------------
```

We could also perform some calculation before typing its result, e.g.

```
F^2-F+1;
```

yields the output

```
x^4 + 2x^2y^2 + y^4 - 3x^2 - 3y^2 + 3
-------------------------------
```

Next we apply one of the built-in CoCoA functions.

```
Factor(F-2xy+1);
```

CoCoA replies

```
[[x - y, 2]]
-------------------------------
```

which means $\mathtt{F} - 2\mathtt{xy} + 1 = (\mathtt{x} - \mathtt{y})^2$.

A.4 Getting Help

A moment ago we saw an example of the use of the CoCoA function Factor(). If you want to know more about this function, type

```
Man('Factor');
```

and CoCoA will display the page of its on-line manual describing the function Factor(). In this case, you may even type

```
Man('fac');
```

Since fac is not the name of a CoCoA command, CoCoA looks for all topics in its on-line manual containing that string. For more extensive information about how to use the manual, type

```
Man();
```

After that, you get a list of possible commands for getting help. For instance, `H.Toc();` will give you the table of contents of the online manual, `H.Tips();` will give you tips for using the online manual, and `H.Commands('Topic');` summarizes all commands associated with `'Topic'` (the quotes are needed to mark a string).

There are several other ways to access the CoCoA manual. In the directory `doc` on the ftp-server mentioned in Section A.1 you will find an html-version for reading with an internet browser, and a postscript-version for printing. Finally, if you use the graphical interface of CoCoA under Windows, you can also access the manual through the Windows help system.

Since this introduction to CoCoA has to be limited in space and scope, we strongly urge you to look up all the topics we discuss in the manual. There you will find more extensive and more detailed information. Moreover, there is a section of the manual called "Tutorial" which complements these appendices and should be studied as well.

A.5 Data Types

Let us get back to our example session. Suppose we now want to define a polynomial ideal. Using the manual, we find that the correct command is `Ideal()` and we can type

```
I:=Ideal(F,x-y);
J:=Ideal(x,y,z)^2;
Intersection(I,J);
```

CoCoA will not reply to the first two lines, since they are assignments. But it reacts to the last line by printing

```
Ideal(xz - yz, xy - y^2, y^2z^2 - 1/2z^2, y^3z - 1/2yz,
y^4 - 1/2y^2, x^2 - y^2)
-------------------------------
```

which means that the intersection of the ideals I and J has been computed. Sometimes it may be advisable to use the command `Set Indentation;` before typing out a list or an ideal. In our case, we'd get

```
Ideal(
  xz - yz,
  xy - y^2,
  y^2z^2 - 1/2z^2,
  y^3z - 1/2yz,
  y^4 - 1/2y^2,
  x^2 - y^2)
-------------------------------
```

Naturally, this behaviour can be reversed by entering `Unset Indentation;`

Many CoCoA objects like polynomials, ideals, and modules are defined over a base ring. The command `CurrentRing();` tells you the value of the current ring. E.g. in our example session we would get the answer `Q[x,y,z]`. A new ring can be defined by a command like

```
Use S::=Z/(5)[x,y,z];
```

Here `::=` is the instruction to create a new ring, `S` is its name, `Z/(5)` is its field of coefficients, and `x,y,z` are its indeterminates. After you have defined the new ring `S`, you are also making it the current base ring by invoking the command `Use <Ringname>;` You can switch back to your initial ring with the command `Use R;` and if you type `RingEnvs();` you get a list of all the base rings defined so far.

Whenever you create a new polynomial, ideal, or module, it is defined over the current ring. To see a list of all objects in the working memory, you can type `Memory();` and to see also the values of those objects you can use `Describe Memory();` At this point you may get a list of descriptions of objects of different types. Some of the most common data types in CoCoA are

`INT, RAT`	arbitrarily large integers resp. rational numbers
`ZMOD`	elements of a finite field $\mathbb{Z}/(p)$ where p is a prime number
`POLY`	polynomials
`IDEAL`	polynomial ideals
`RING`	polynomial rings
`VECTOR`	vectors whose entries are polynomials
`MODULE`	submodules of finite free modules over polynomial rings
`LIST`	Lists of objects of the form `[Object1, Object2, Object3]`
`MAT`	matrices of polynomials
`STRING`	strings (sequences of characters)
`BOOL`	Boolean variables (i.e. their value is either `TRUE` or `FALSE`)

In CoCoA, you usually do not have to specify the data types of the objects you create. Most sensible data type conversions are done automatically. The command `Type(Object)` displays the data type of an `Object`. If necessary, you can also force type conversions using `Cast()` e.g.

```
L:=[x,y,z];
Type(L);
```

shows that `L` is of type `LIST`, whereas

```
Cast(L,IDEAL);
```

leads to the answer

```
Ideal(x, y, z)
-------------------------------
```

For special type conversions, there are frequently individual commands, e.g. the above conversion could have been done by typing `I:=Ideal(L);` Then the ideal `I` can be converted back to a list by using `Gens(I);` which brings us to our next topic: What is a list?

A.6 Lists

In CoCoA, a list is a sequence of objects, separated by commas and enclosed in square brackets. Lists are very common and useful objects in CoCoA. Examples of lists are for instance `[1,2,3]` and `[x,y,z]`. The different components of a list do not necessarily have to have the same data type, e.g. as in the list `[1,TRUE,x]`. Given a list `L:=[x,y,z];` you can access its second component by typing

```
L[2];
```

which yields the result `y`. Another way to achieve the same result is to type `Comp(L,2);` This is especially convenient if the list `L` is not directly given by its name but the result of some other evaluation, e.g. as in `Comp(Gens(I),1);` where `I` is an ideal.

Besides listing all the elements, there are other ways to create a list. Two simple ones are the range operator and the Cartesian product. The range operator `N..M` creates the list of all integers between `N` and `M`. For instance,

```
1..5;
```

creates the list

```
[1, 2, 3, 4, 5]
-------------------------------
```

The range operator can also create a list of indeterminates of the current ring (in the current term ordering), e.g. if the current ring is `Q[x,y,z]`, then `x..z` creates the list `[x,y,z]`, and if the current ring is `Q[x[1..8]]`, then `x[1]..x[4]` creates the list `[x[1],x[2],x[3],x[4]]`. The Cartesian product of two lists is the list of all pairs, e.g. the product `[1,2,3] >< [x,y]` yields the list `[[1, x], [1, y], [2, x], [2, y], [3, x], [3, y]]`.

There are many built-in CoCoA function for dealing with lists. We invite you to print a list of them using `Man('Functions for Lists')`; In our opinion, some of the most useful ones are

a) `Append(L,O)` to append an object `O` to a list `L`,
b) `Concat(L,M)` to combine two lists `L` and `M` into one,
c) `First(L,N)` and `Last(L,N)` to retrieve the list of the first and last `N` elements of a list `L`, respectively,
d) `Diff(L,M)` to create the list of all elements of the list `L` which are not in the list `M`, and
e) `Len(L)` to determine the number of elements of a list `L`.

If you want to build a new list by computing (or entering) its elements one by one, you can either apply commands such as `Append(L,x);` repeatedly, or you can first define a new "empty" list of a specified length `N` via `L:=NewList(N);` and then assign its entries using commands such as `L[1]:=x;`

Notice that, unlike other functions for lists, the command `Append(L,x)` does not return a value, but modifies the argument list `L`. CoCoA tries to be clever about operations with lists. For instance, if `L:=[x,y,z];` then `2L;` yields the list `[2x, 2y, 2z]`. But of course, it has to leave some thinking to you, and a construction like `[x,y,z]^3;` makes no unique sense.

Finally, we note that more advanced ways to construct lists will be discussed in Appendix B.5.

A.7 It

Finally, we would like to mention a special CoCoA variable called `It`. At each point in time it contains the last output. In particular, if you print the result of a computation and then discover that you still need it, you can assign it to a variable using `It`, e.g.

```
Factor(x^9-1);
[[x - 1, 1], [x^2 + x + 1, 1], [x^6 + x^3 + 1, 1]]
-------------------------------
L:=It;
L;
[[x - 1, 1], [x^2 + x + 1, 1], [x^6 + x^3 + 1, 1]]
-------------------------------
```

Of course, we encourage you to develop a habit of always assigning the result of a computation to a variable, e.g. using `L:=Factor(x^9-1);` and then to print it. As one CoCoA programmer put it:

Never use `It`!