CS1110 Introduction to Systematic Programming

Emacs Ada-Mode[1].

There are a number of **modes** which affect the way Emacs behaves. There are major modes for most computer languages: `ada-mode`, `c-mode`, `html-mode` and many others; these provide special features which are useful for editing computer programs in that language.. The default mode for Emacs is `text-mode` which is suitable for editing general text files.

If a file with the extension `.adb` or`.ads` is edited with Emacs, Ada mode is entered automatically. Ada-mode affects the behaviour of Emacs as program text is entered. **Try out the features of Emacs Ada-mode the next time you edit an Ada program in the lab classes**.

Automatic Case Adjustment

Ada keywords are automatically changed to upper-case after they have been typed. Similarly the leading letter of identifiers is capitalised automatically. This is the recommended style for the ISP module.

The case adjustment feature can sometimes help in the detection of typing errors: for example suppose the keyword `LOOP` is mis-spelt with two `0`'s (zeros) instead of `O`'s (capital oh), then `L00p` will be treated as a identifier and not a keyword and so only the first letter will be capitalised. These visual cues should alert you that sometime is wrong whereas without them it is often extremely difficult to detect the difference between a `0` and an `O` on some computer monitors.

Similarly if you accidentally use an Ada keyword as an identifier then Emacs Ada mode will convert the whole word to capitals and so alert you to the fact that your choice of identifier is invalid -- choose an alternative name. For example suppose you were to try to use `Case` as an identifier, Emacs would convert this to `CASE` after you had typed it (because it is an Ada keyword -- see Unit 14), you should choose something else, say `Valise` or `Suitcase`!

Automatic Line Indentation

Pressing `<Tab>`[2] automatically adjusts the indentation of the current line of program code so that it lines up relative to preceding program lines in the recommended ISP style.

It is important that program are properly indented to improve program clarity by emphasizing the logical structure and so aiding human understanding of the program. As you type in your programs, make a habit of pressing the `<Tab>` key at the end of each line to correctly indent your program. Note that auto-indentation works by aligning the current line relative to <u>earlier lines</u> and so it is important to start indenting correctly from the start of the file.

As well as keeping lecturers happy by correctly indenting your coursework submissions, auto-indent can also help detect programming errors, for example if a semi-colon is missed out on the previous line, the current line will be regarded as a continuation line and will be indented in a different way from the usual. These visual cues should alert you to the fact that something is wrong and enable you to find and correct many common typing errors.

Syntax Colouring

Syntax colouring is useful feature of many Emacs modes. In Ada-mode

Ada keywords	are displayed in	purple
subprogram names		blue
comments		red
strings		pinkish brown
type names		green
other Ada text		black

This gives additional visual clues to aid in the detection of syntax errors, for instance if an intended keyword (`END IF`) is mis-typed (as `ENDIF`, say) it will not be recognised as a keyword and so will be displayed in black rather than purple. This may aid detection of such typos.

---

[1] The features described in this hand-out will only work as described if you correctly set up your Unix environment as described in the labs in week 1.

[2] The tab key is sometimes marked as →|.

Similarly if the closing double quotes in a string are omitted in a line such as

```
Put(Item => "Please input an integer); New_Line;
```

the final portion of the line: `); New_Line;` will be regarded as part of the string and so will be displayed in pink rather than black. The additional visual colour cues should help the detection of typos at an early stage.

A number of useful commands are available from the special Ada-mode menu **Ada**. We have discussed use of **Build**, **Compile** and **Run** which allow a program to be compiled, bound, linked and run from within Emacs in a previous hand-out

The Ada Statements Sub-Menu
Templates for various Ada constructs, `WHILE`, `FOR`, `IF`, `ELSE`, `ELSIF` and `CASE` etc. are available from the **Statements** sub-menu of the **Ada** menu. For example selecting 'while loop' inserts the template

```
WHILE <condition> LOOP

END LOOP;
```

in the edit buffer at the current cursor position point. The user is prompted for the loop entry `<condition>` in the mini-buffer[3]. For example if you type `Data /= Terminator` in response and press `<Return>` the loop template inserted will become

```
WHILE Data /= Terminator LOOP

END LOOP;
```

If you are unsure about the precise sysntax of a particular Ada command, using a template from the menu should help you get the syntax correct.

The Ada Edit Sub-Menu
Other Ada-specific editing commands are available on the **Edit** sub-menu of the **Ada** menu. These include:

`Indent Lines Selection`	indents the lines of a selected region
`Adjust Case Selection`	adjust case of keywords, identifiers etc. in selected region
`Comment Selection`	adds `--` at the start of each line of the selected region
`Uncomment Selection`	removes `--` from each line of the selected region
`Fill Comment Paragraph`	formats multi-line comments nicely
`Format Parameter List`	formats a parameter list in a subprogram heading

To select a region simply drag over it with the mouse and then select a command from the menu to act on the region. To format a comment or parameter specification just click the mouse on the required line and choose the formatting command from the menu. There are also commands for indenting and case-adjusting all the lines in a file

The Ada GoTo Sub-Menu
This sub-menu contains some useful commands for moving about in an Ada program:

`Goto Start`	if the cursor is on an `END`, find the matching `WHILE`, `IF` etc.
`Goto End`	finds the `END` of a compound statement
`Next procedure`	move cursor to next procedure declaration
`Previous procedure`	move cursor to previous procedure declaration
`Next compilation errror`	move cursor to next compilation error

The first two commands are useful for detecting missing/misplaced `END IF`, `END LOOP` commands etc.. The next two commands are useful for moving between procedure declarations (see Units 7 & 8 for information on defining your own procedures). The fifth command is useful in correcting compilation errors in a file (of course this only works after compiling the file by selecting **Build** from the **Ada** menu). Note you can also move between compilation

[3] You are also prompted for the loop name, but as we do not use named loops in the ISP course simply press <Return> to get an unnamed loop.

errors by selecting **Next Error** or **Previous Error** from the **Compile** menu in the compilation buffer that appears when a file is compiled from within Emacs.

Emacs Quick Keys
The normal way of invoking Emacs commands is via the various menus at the top of the Emacs window. This is useful for novice users, however it can be rather slow as one hand must be removed from the keyboard to move the mouse. All Emacs commands can be invoked by a few key-strokes. The most commonly used Emacs 'quick keys' involve control characters which are typed by pressing a key with the `<Control>` key held down. For brevity we denote the control characters `<Control-a>` and `<Control-e>` by `C-a` and `C-e` and so on. Here are a few useful quick-keys.

`C-a`	Move cursor to start of current line
`C-e`	Move cursor to end of current line
`C-v`	Scroll down one page
`C-d`	Delete character under the cursor
`C-k`	Delete to end of line (second `C-k` deletes the empty line so formed)
`C-x C-s`	Save buffer to file
`C-x C-w`	Write buffer to a new file
`C-x C-c`	Quit Emacs

Note that `C-d` deletes the character under the cursor whilst `<Delete>` deletes the character before the cursor. Hence repeatedly pressing (or holding down) the `<Delete>` deletes **backwards** from the current point **whilst** repeatedly pressing (or holding down) `C-d` deletes **forwards** from the current point.

Other quick sequences are invoked with the **meta-key** (usually the same as the escape key). For example `<Meta-x>` or `M-x` for short is invoked by pressing the `<Escape>` key followed by the `x` key.

`M-v`	Scroll up one page
`M->`	Scroll to end of buffer
`M-<`	Scroll to start of buffer
`M-f`	Move to end of current word
`M-b`	Move to start of current word
`M-d`	Delete current word
`M-<Delete>`	Delete previous word

Note that many quick key sequences appear on the Emacs menus next to the correspponding command. For a full listing of all quick key bindings (several hundred) select **Describe Key Bindings** from the Emacs **Help** menu.

Repeating Commands
Some times it is useful to repeat an edit command several times, this can be done by giving the command a numerical argument with `C-u`. For example

`C-u 6 M-f`	move forward six words
`C-u 10 M-<Delete>`	delete previous 10 words
`C-u  M-b`	move back four words (if no numerical argument given, the default value of 4 is used)
`C-u C-u M-b`	move back sixteen (=4*4) words
`C-u 60 -`	type a line of exactly 60 hyphens (useful for sectioning up Ada code)

Changing Emacs Major Modes Manually
Usually Emacs automatically enters the correct mode corresponding to the type of the file being edited and so it is not usually necessary to change mode manually. However occasionally you may wish to change mode from the default. To do this is to type `<ESC> X`, Emacs will prompt you in the mini-buffer with `M-X`. Now type the mode name in the Emacs mini-buffer, for example

```
M-x ada-mode            M-x text-mode     M-x c-mode
```

In fact all Emacs commands can be entered by typing `<ESC>x` followed by the command name.

Minor Modes

Emacs modes may either be **major** or **minor**; only one major mode may be active in a given buffer at one time and this may be supplemented by one or more minor modes. Minor modes modify the behaviour of Emacs in less fundamental ways than major language modes.

`line-number-mode` causes Emacs to display the current line number of the cursor on the status line. By default Ada mode also starts `line-number-mode`.

`auto-fill-mode` is mainly useful in `text-mode`. It affects whether words are wrapped at the end of a line (i.e. whether a word which won't fit onto a line is moved to the next line or whether the line is extended beyond its normal width).

`overwrite-mode` affects what happens when characters are inserted (by typing) in the middle of an existing line. The default is `overwrite-mode` off: characters are inserted as they are typed and the rest of the line is shifted to the right to make room. In `overwrite-mode` existing characters are overwritten one by one as you type. Usually overwrite-mode is best avoided.

Changing Emacs Minor Modes

You can toggle[4] the states of `auto-fill-mode` and `line-number-mode` by doing

```
            M-x auto-fill-mode            M-x line-number-mode
```

as appropriate. To change from normal `insert-mode` to `overwrite-mode`:

```
            M-x overwrite-mode
```

To turn off `overwrite-mode`, the command must be given a negative argument:

```
      C-u -1 M-x overwrite-mode
```

where `C-u` means `Control-u`, that is press the `Control` and `u` keys together.

Emacs Automatic Back-up Files

As a safety measure, Emacs retains the previous version of any file that you edit. This is useful if you completely mess up an edit and make the mistake of saving the changes; you still have a copy of the original version safely stored on disk. The original file is saved as an **automatic back-up**. The name of the back-up file is obtained by appending a tilde (~) to the original file name. For example if the original file was called `prog1.adb`, the back-up will be called `prog1.adb~`. Note if you completely mess up an edit, rename the backup file to be the original file name

```
      mv prog1.adb~ prog1.adb
```

and then edit the file -- don't edit the automatic back-up file directly with Emacs.

Emacs Auto-Save Files

As an extra insurance, as you edit a file, Emacs always creates (and updates every few minutes) an **auto-save file** so that if the system crashes in the middle of a long editing session you only lose a few minutes work. The name of the auto-save file is automatically generated by Emacs by adding the hash character `#` to the front and end of the filename. For example the auto-saved version of `prog1.adb` is called `#prog1.adb#`. If Emacs exits normally the auto-save file is deleted, but if Emacs exits abnormally (due perhaps to a system crash or a quit without save) the auto-save file will be retained.

Again if you want recover a file after a system crash, rename the auto-save file to have its original name

```
      mv #prog1.adb#  prog1.adb
```

Then edit the file in the normal way. Beware editing the auto-save file directly with Emacs as this turns off the auto-save protection!

[4] Toggle:- if the state is on, change it to off, whereas if the state is off, change it to on.