

Introduction to Emacs

Bob Dowling

rjd4@cam.ac.uk

15 May 2006

Contents

Introduction.....	3
Course outline.....	3
Setting up some demo files.....	3
What those commands did.....	3
Launching Emacs.....	4
Quitting Emacs and undoing edits.....	6
Quitting.....	6
Notation.....	6
Undo.....	6
Summary of commands.....	6
Loading a file.....	7
Summary of commands.....	10
Moving around in a file and making trivial edits.....	11
Summary of commands.....	13
Editing C files.....	14
Removing the comment.....	14
Adding a line above.....	15
Adding the if statement.....	15
Adding the opening brace.....	16
Indenting the fprintf() line.....	17
Adding the closing brace.....	17
Setting up the debug variable.....	18
Editing Fortran files.....	21
Removing the comment.....	21
Adding a line above.....	22
Adding the IF statement.....	23
Indenting the WRITE line.....	23
Adding the ENDIF statement.....	24
Setting up the DEBUG variable.....	25
Editing Python files.....	28
Removing the comment.....	29
Adding a line above.....	29
Adding the if statement.....	30
Indenting the print line.....	30
Bracket matching.....	30
Adding a line below.....	31
Setting up the debug variable.....	32
Switching between buffers.....	33
Summary of commands.....	35
Searching and replacement.....	36
Summary of commands.....	43
Jumping to a line.....	44
Summary of commands.....	48
Copying, Cutting, and Pasting.....	49
Summary of commands.....	51
Keystroke Macros.....	52
Summary of commands.....	55
Recap and conclusion.....	56
Summary of key strokes.....	57

Further reading.....	58
----------------------	----

Acknowledgements

Emacs was originated by Richard Stallman, is © the Free Software Foundation¹ and is released under the terms of the GNU General Public Licence.

The text of Treasure Island used in the examples was written by Robert Louis Stephenson and was made available on-line by Project Gutenberg².

Warning

These notes make extensive use of colour. People printing them in greyscale may have problems.

1 <http://www.fsf.org/>

2 <http://www.gutenberg.org/>

Introduction

This is a course on the basic use of the Emacs editor. The platform used is PWF Linux but the principles and techniques should apply to all systems. The only common version of Emacs with dramatically different key bindings is Aquamacs for MacOS X which follows the MacOS X standard. Some twisted system administrators do impose their own particular key bindings on an Emacs installation but these people are evil and probably torture puppies for fun³.

The presumption is that Emacs will be running in a graphical environment. Emacs can be used in a text console and much of what is taught moves directly over, but it will not be covered in this course.

Course outline

This course is designed to give just the basic grounding in Emacs. The most commonly used options will be demonstrated and the support Emacs offers to programmers in various languages will be illustrated.

The notes cover this assistance in various languages. In the lecture a specific one will be chosen by vote of the audience. The principles apply fairly uniformly, though, and once you get the hang of the Emacs way of doing things, moving between languages is easy.

Setting up some demo files

We have some demo files to work with. Launch a terminal window and type the following commands:

```
$ mkdir EmacsCourse
$ cd EmacsCourse
$ cp /ux/Lessons/Emacs/demofiles/* .
$ ls -l
total 380
-rw-r--r--  1 rjd4 rjd4   2230 2006-02-24 14:00 iterator1.c
-rw-r--r--  1 rjd4 rjd4    693 2006-02-24 15:55 iterator2.f
-rw-r--r--  1 rjd4 rjd4    418 2006-02-24 16:00 Makefile
-rw-r--r--  1 rjd4 rjd4    102 2006-02-24 15:58 temp.gplt
-rw-r--r--  1 rjd4 rjd4 364748 2006-02-24 13:54 treasure.txt
$
```

This will give us something to play with.

What those commands did

`mkdir EmacsCourse`: Make a directory called “EmacsCourse” which we will use to store the files we will practice on.

`cd EmacsCourse`: Move into the EmacsCourse directory.

`cp /ux/Lessons/Emacs/demofiles/* .`: Copy all the individual files from the /ux/Lessons/Emacs/demofiles directory into your current directory. On a Unix or Linux system, the name “.” always means “the directory you are currently in”.

`ls -l`: Give a long format listing of the current directory.

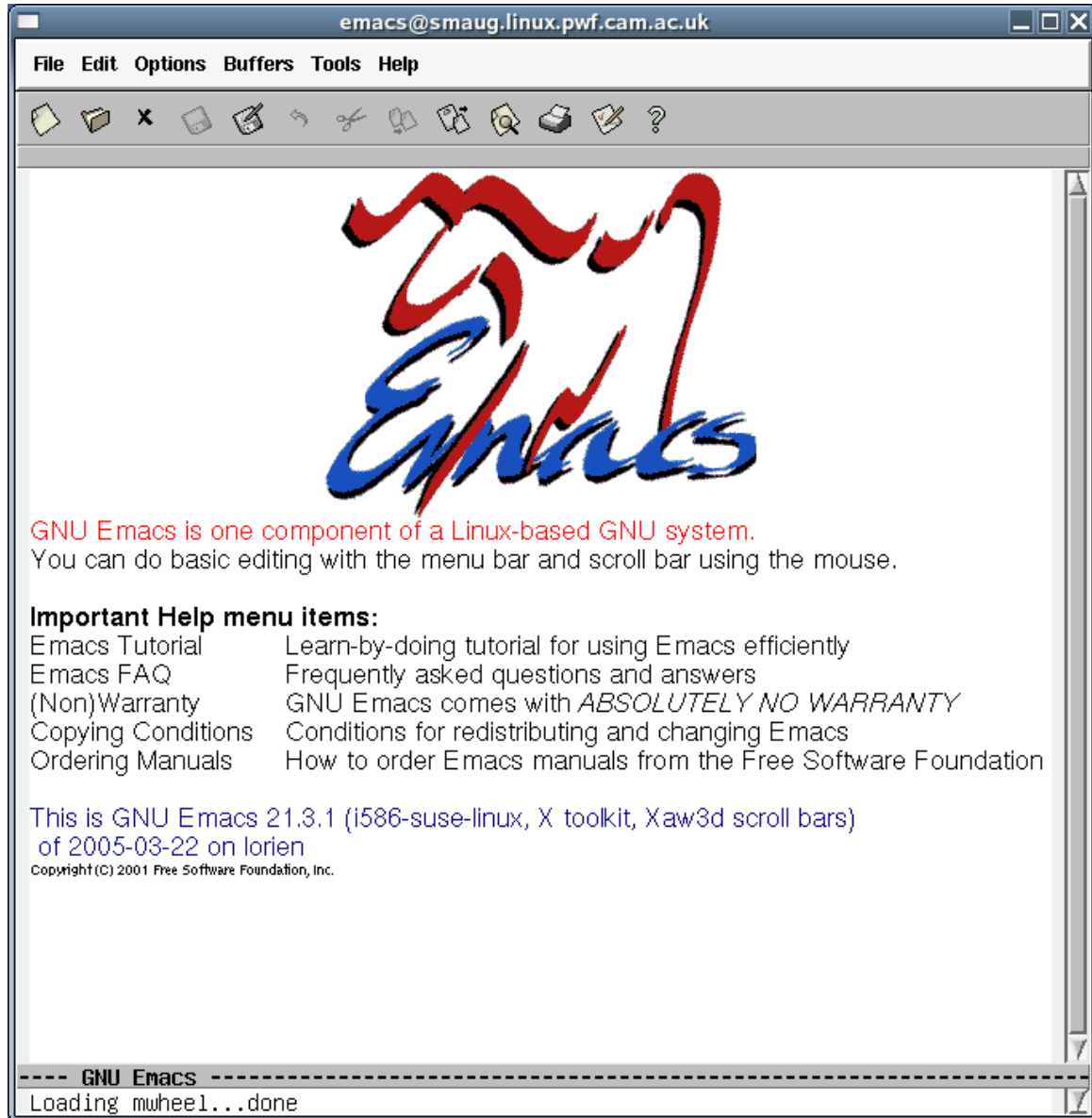
³ They are also, mercifully, quite rare.

Launching Emacs

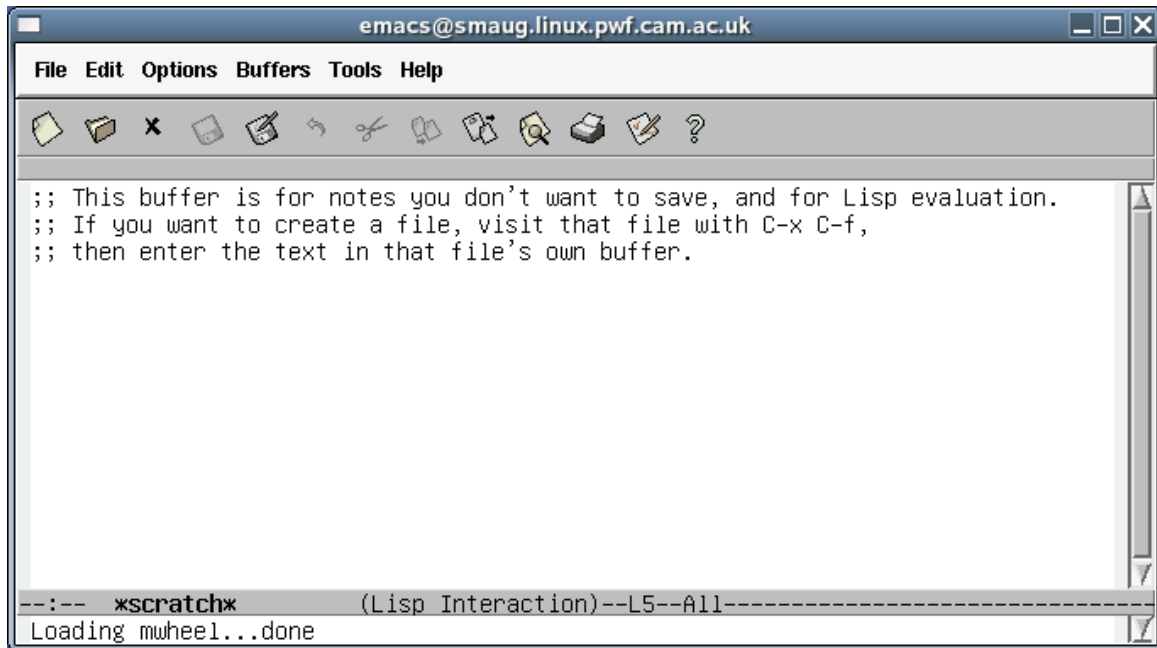
Issue the command

```
$ emacs &  
$
```

and a copy of Emacs should start running. The command “emacs” is simply the name of the command that launches the Emacs editor and the trailing ampersand, “&”, starts it up “in the background” so that you can carry on using the the terminal while Emacs is still running.



When freshly launched, Emacs should appear as above, with your own machine name in the title bar. If you click in the window or just leave it alone for long enough this will be replaced with something else:



We will use this to study the anatomy of an Emacs window. We will start at the top:

- The very top is the title bar, of course, identifying this as Emacs running on a machine called `smaug.linux.pwf.cam.ac.uk`. (Some of the screenshots are taken from Emacs running on other systems. Don't panic; it's just that it takes time to write notes like this.)
- Below that is the menu bar. Of immediate interest to us is that the Help menu has an "Emacs tutorial" as one of its menu items.
- Beneath that is a bar of short cuts to various operations. We're not going to use these at all, preferring the keyboard to issue commands, but they exist for the point and click brigade.

Then there is the text currently being processed by Emacs:

```
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.
```

At the bottom of the window are two lines. We will start with the first line:

First it identifies the thing being edited (called a "buffer" in Emacs) as having the name `*scratch*`.

It also specifies the way Emacs will process what's in the buffer. "Lisp Interaction" means that it is expecting the file to contain entries in the Lisp programming language (which underlies Emacs).

The "L5" entry says that the cursor is currently on line 5 of the buffer.

The "All" item says that the whole buffer is being displayed.

The final line as shown says `Loading mwheel...done`. This is where status messages will appear and where we will type some commands. It is called the "mini-buffer".

Quitting Emacs and undoing edits

Quitting

The flip side of knowing how to launch Emacs is knowing how to quit it.

Press Control+X followed by Control+C. By Control+X we mean hold down the Control key and then hit the X key. When performing a sequence of control keys like this you can hold down the Control key, hit the X key, hit the C key, and then release the Control key.

Notation

Within the Emacs documentation there is a particular notation for Control keys which we will use here too. Control+X followed by Control+C is written “C-x C-c”.

Later we will meet Control+X followed by simple (non-control) keys. Control+X followed by the letter B is written “C-x b”.

We will also meet another modifier similar to the Control key. On a PC keyboard this is typically marked as the Alt key but its Emacs name (the “Meta” key) dates back before it had that label. Alt+X is written “M-x” in Emacs as a result. But we don't need to worry about that yet.

Undo

This is also a useful time to mention how to “undo” in Emacs. To undo the most recent editing operation, press C-_⁴. Repeated use of this will wind back the editing history. Emacs has a very long editing history.

Finally, we will be seeing some relatively drawn out Emacs commands. Defining a region to cut or copy can last a while, as can defining a keyboard macro. If you want to abandon any particular long operation, press C-g to quit it.

Summary of commands

At the end of each section of the notes there will be a quick reminder of the commands met in the section. They will all be summarized in an appendix at the end.

Key sequence

C-x C-c

C-_

C-g

Operation

Exit Emacs

Undo last edit

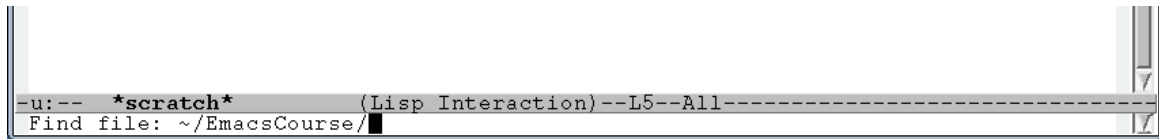
Quit current command

⁴ Control underscore. Underscore is a shifted character on many keyboards (Shift+hyphen on UK PC keyboards).

Loading a file

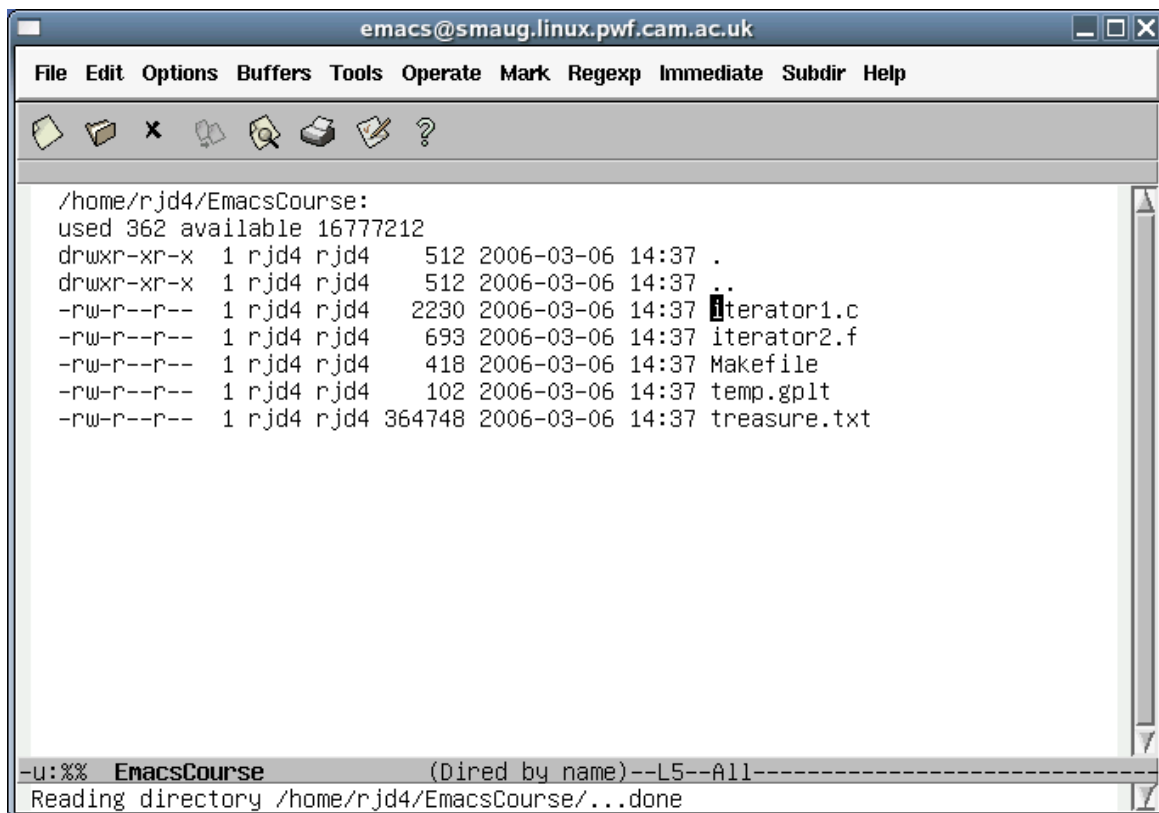
Type C-x C-f.

The mini-buffer will change to look like this:



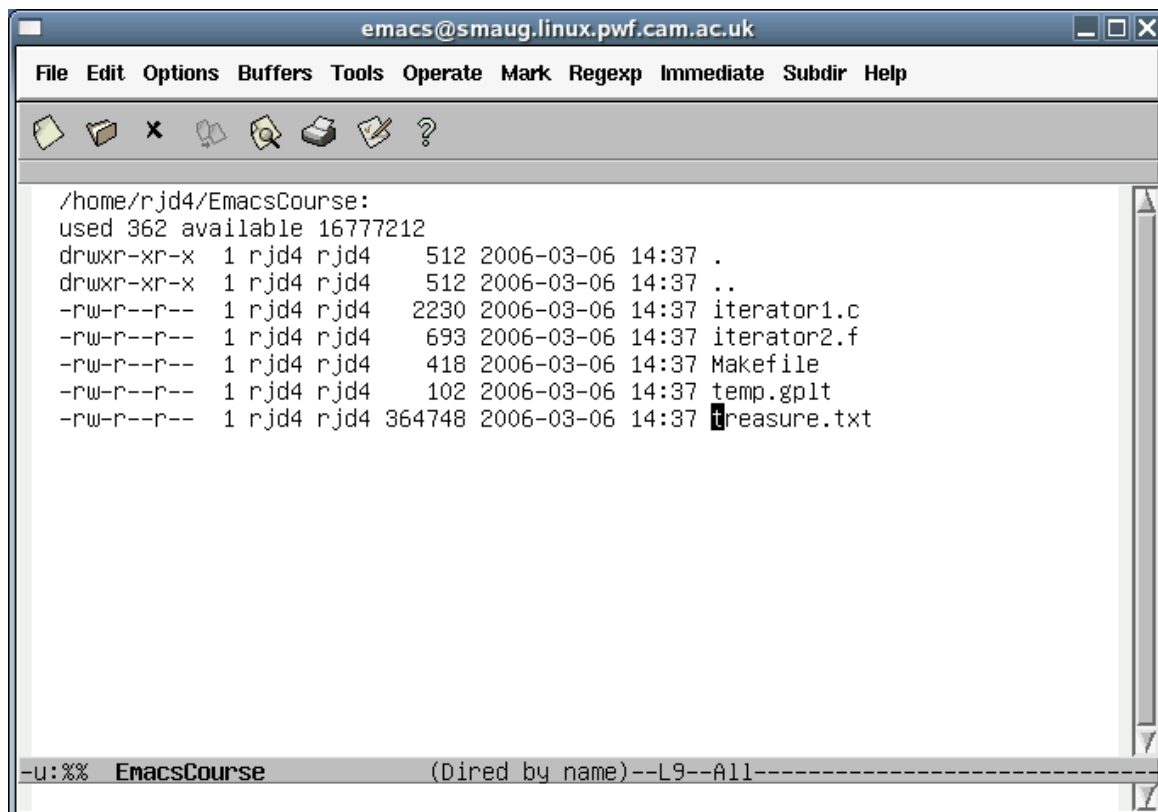
We have a choice at this point. We can hit Return. This will cause Emacs to open the directory for our viewing pleasure. Alternatively we can type a file name in the directory for Emacs to use.

We will start with the former case: opening the directory. If we simply hit the Return key then the editing buffer shows the content of the directory as shown by the command "ls -l":



The mini-buffer shows the success of the operation.

Notice how the cursor in the Emacs buffer is over the initial "i" of `iterator1.c`. If we use the up and down arrow keys we can move the cursor to other files. For example we can move all the way down to `treasure.txt`:

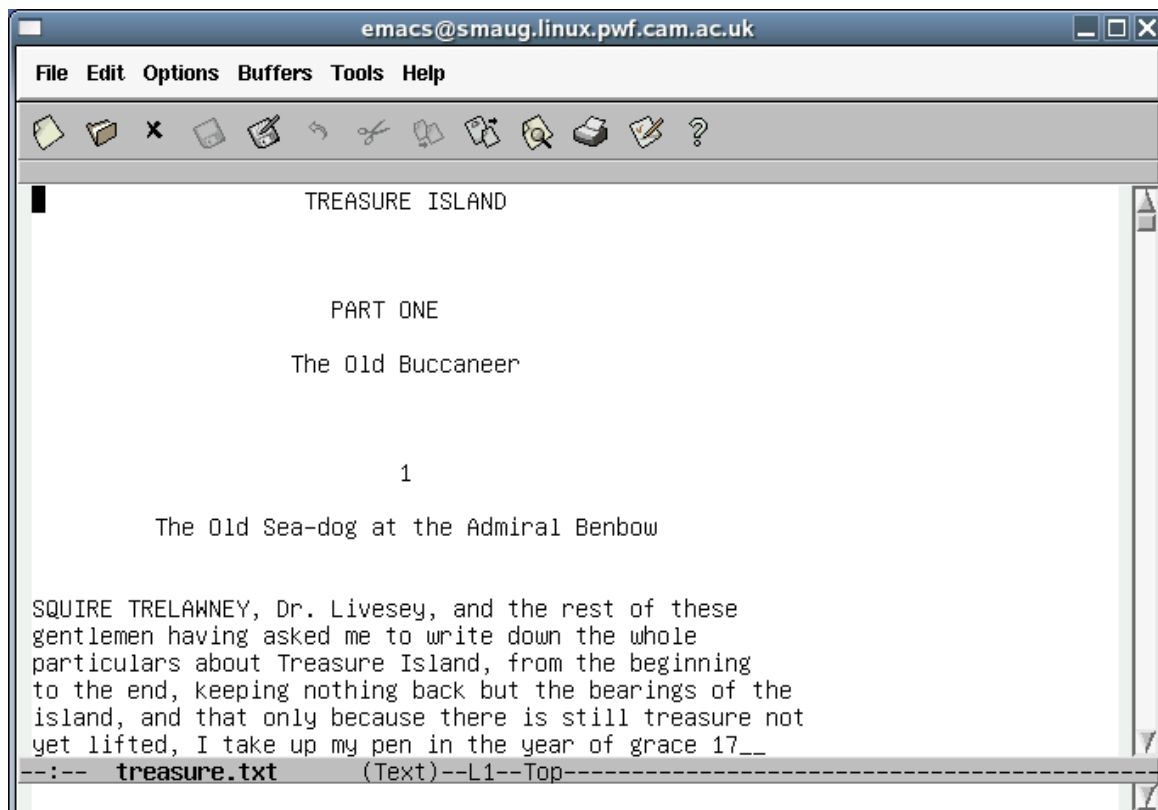


The screenshot shows the Emacs window titled `emacs@smaug.linux.pwf.cam.ac.uk`. The menu bar includes `File Edit Options Buffers Tools Operate Mark Regexp Immediate Subdir Help`. The toolbar contains icons for file operations. The main buffer displays a directory listing for `/home/rjd4/EmacsCourse:` with the following content:

```
used 362 available 16777212
drwxr-xr-x  1 rjd4 rjd4    512 2006-03-06 14:37 .
drwxr-xr-x  1 rjd4 rjd4    512 2006-03-06 14:37 ..
-rw-r--r--  1 rjd4 rjd4   2230 2006-03-06 14:37 iterator1.c
-rw-r--r--  1 rjd4 rjd4    693 2006-03-06 14:37 iterator2.f
-rw-r--r--  1 rjd4 rjd4    418 2006-03-06 14:37 Makefile
-rw-r--r--  1 rjd4 rjd4    102 2006-03-06 14:37 temp.gplt
-rw-r--r--  1 rjd4 rjd4  364748 2006-03-06 14:37 treasure.txt
```

The status bar at the bottom shows `-u:%% EmacsCourse (Dired by name)--L9--All-----`. The cursor is positioned at the start of the `treasure.txt` line.

While the cursor is on any file's row we can hit Return again and open that file:



The screenshot shows the Emacs window titled `emacs@smaug.linux.pwf.cam.ac.uk`. The menu bar includes `File Edit Options Buffers Tools Help`. The toolbar contains icons for editing and file operations. The main buffer displays the contents of `treasure.txt`:

```
TREASURE ISLAND

PART ONE

The Old Buccaneer

1

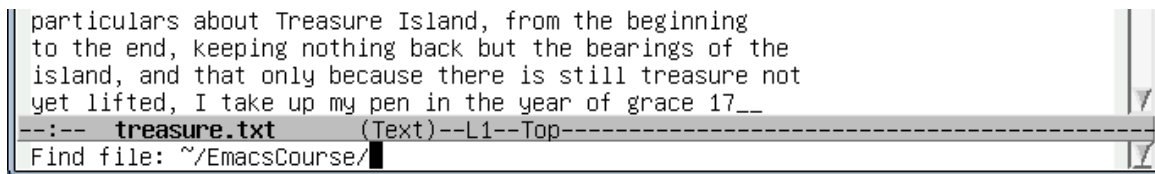
The Old Sea-dog at the Admiral Benbow

SQUIRE TRELAWNEY, Dr. Livesey, and the rest of these
gentlemen having asked me to write down the whole
particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
```

The status bar at the bottom shows `--:-- treasure.txt (Text)--L1--Top-----`. The cursor is at the beginning of the first line.

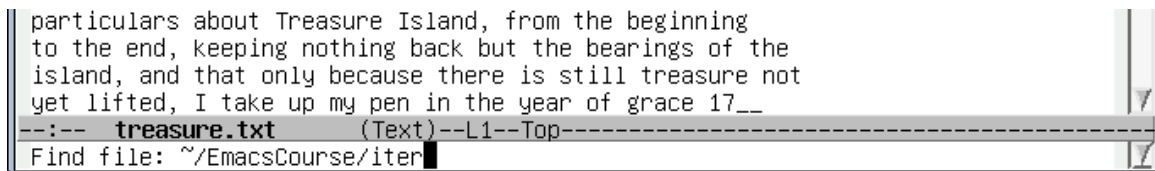
We now have a file open for reading.

The other way to open a file is to enter its full name in the mini-buffer after C-x C-f:



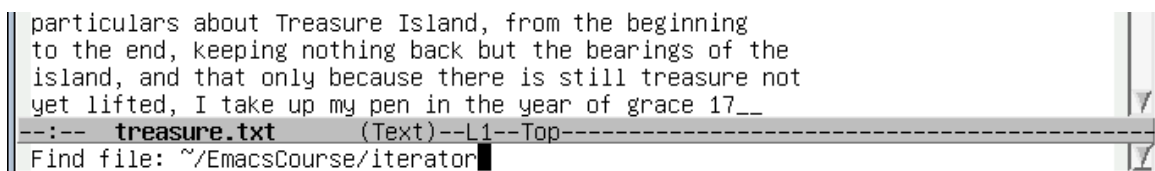
```
particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Find file: ~/EmacsCourse/
```

and then start to type a file name: "iter":



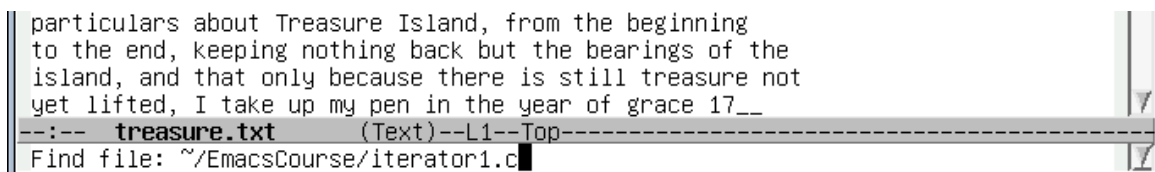
```
particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Find file: ~/EmacsCourse/iter
```

and then press the Tab key. The file name in the mini-buffer is extended as far as Emacs can work out what it should be, namely "iterator". Emacs doesn't know whether iterator1.c or iterator2.f is wanted.



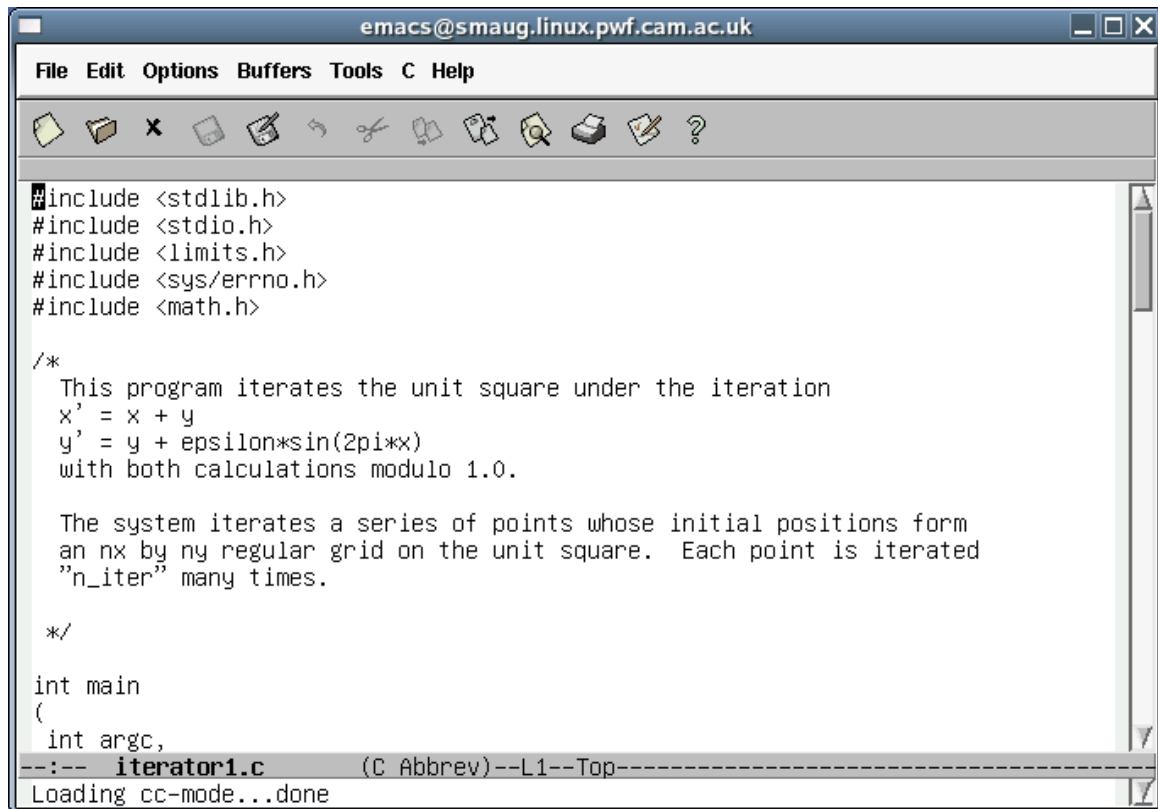
```
particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Find file: ~/EmacsCourse/iterator
```

We can press "1" and Tab again at this point to complete the file name:



```
particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Find file: ~/EmacsCourse/iterator1.c
```

Press return to open the file.



Summary of commands

Key sequence

C-x C-f

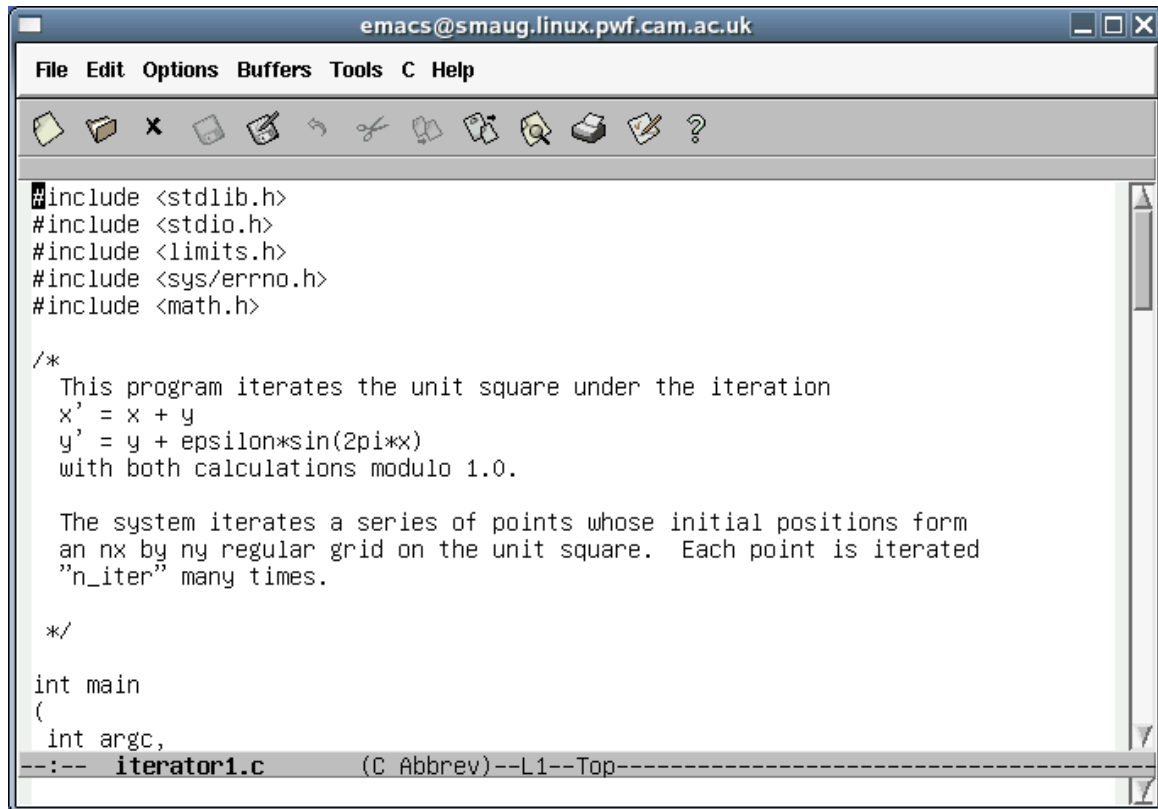
Operation

Open a file

Moving around in a file and making trivial edits

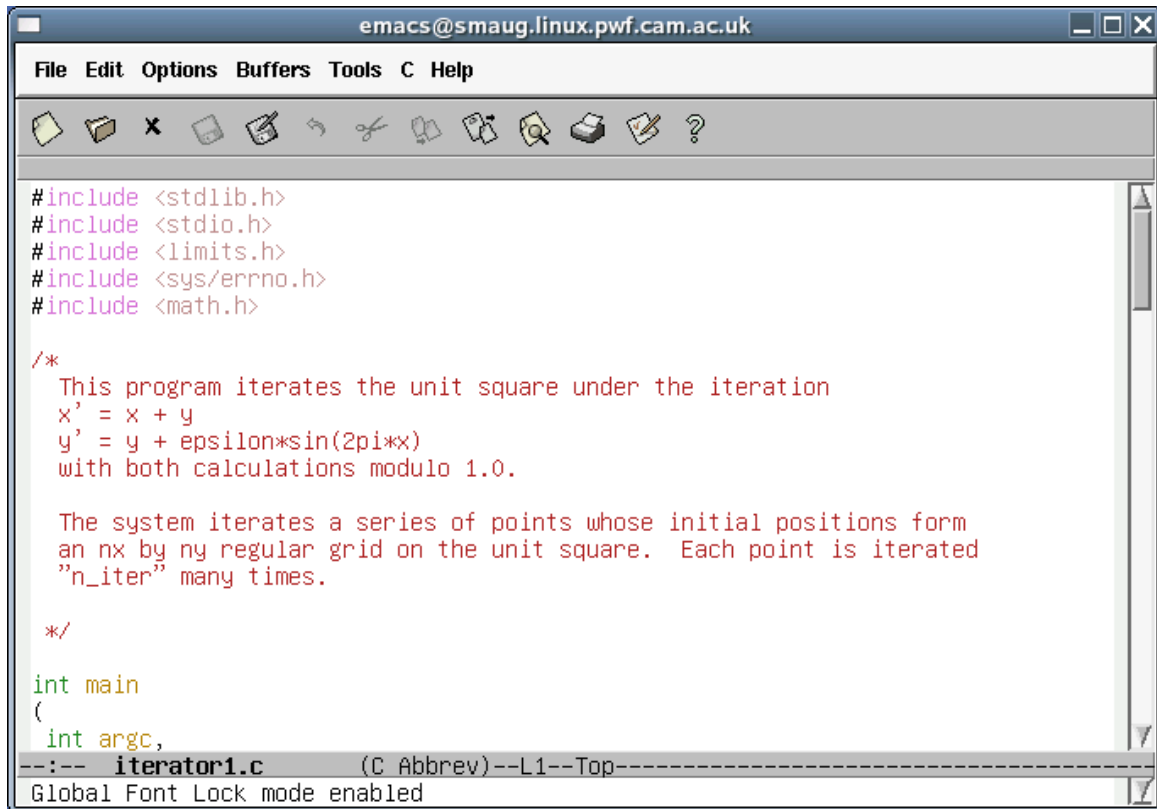
Now we have files loaded into Emacs we want to edit them.

We will select the `iterator1.c` buffer. While this section uses C, you don't need to be a C programmer to understand the Emacs used. In fact, all you need to know about C is that everything between `/*` and `*/` is a comment and is ignored.



Note how the line near the bottom contains the text `"(C Abbrev)"`. This means that the editor knows it is editing a C program and is in "C mode". Certain operations are configured to be useful to people writing C.

To see this in glorious technicolour⁵, use your mouse to select the “Options” menu from the menu bar at the top. The top option is “Syntax highlighting (Global font lock mode)”. Select it. You will find that your C program is now colourized according to the rules of C syntax:



Regrettably, people printing out these notes in grey scale will not get the full lurid effect of this operation. Later we will see how to do the same operation from the keyboard.

We will start with an absolutely trivial change. We can use the cursor keys (Up, Down, Left, Right⁶) to move the cursor to the start of the line containing the comment

```
y' = y + epsilon*sin(2pi*x)
```

and across to the first “*” of the line. This comment is actually incorrect as the line should say

```
y' = y + epsilon*y*sin(2pi*x)
```

We can simply type “*y” here to insert text to the left of the cursor.

Similarly we can change the lead line of the comment from

```
This program iterates the unit square under the iteration
```

to

```
This program iterates the unit square according to
```

by moving the cursor to the end of the line and pressing the BackSpace key until the offending text is removed and then typing in our replacement. (The Delete key deletes the character currently under the cursor and all the text to the right slides one place to the left to fill in the gap.)

Note that we can move to the beginning or end of the current line by pressing the Home or End keys (or by pressing C-a or C-e, which also work on systems without PC style keyboards).

- 5 Unless you are colour blind. Nothing in this course, or in the use of Emacs, relies on being able to see colour. Emacs uses it for hinting and suggestion. In this case it uses it to hint about your syntax.
- 6 If you ever find yourself on a keyboard without the cursor keys (and a few do still exist) then there do exist control key equivalents: C-p: Up/**P**revious line, C-n: Down/**N**ext line, C-f: Left/**F**orward, C-b: Right/**B**ackward.

After making any substantive change it is important to *save your work*. Now Emacs can help you here by making its own backup copies as it goes along but you really should get into the habit of saving back to your original document as you go. To save your work, press C-x C-s.

We can move further than a one character or row at a time. To move to the very end of the buffer, press the Alt key and the ">" ("greater than") key. Note that the ">" key is typically a Shift character in its own right⁷. Also note that the Alt key was originally called the "Meta" key and this name has stuck in the Emacs notation. Alt+> is written M-> for "meta greater than". Similarly M-< will take the cursor to the very start of the buffer. Also note that the "AltGr" key is *not* the same as the Alt key.

We can also move about a screenful at a time. The PageDown and PageUp keys (with equivalent control sequences C-v and M-v) move one screenful at a time down or up the buffer.

Summary of commands

Key sequence	Short cut	Operation
C-b	Left	Move cursor left (b ackwards) one character
C-f	Right	Move cursor right (f orwards) one character
C-n	Down	Move cursor down one row (to n ext row)
C-p	Up	Move cursor up one row (to p revious row)
C-a	Home	Move cursor to start of line
C-e	End	Move cursor to e nd of line
C-x C-s		Save the file
M-<		Move to start of buffer
M->		Move to end of buffer
C-v	PageDown	Move one screenful down the buffer
M-v	PageUp	Move one screenful up the buffer

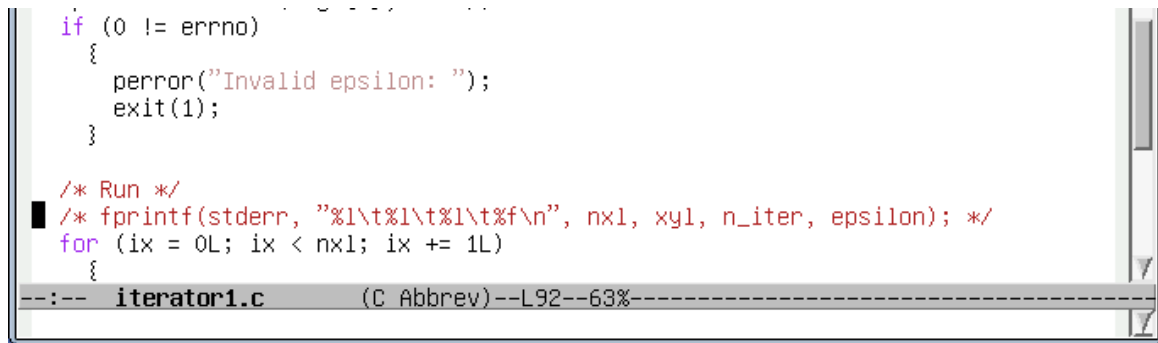
⁷ ">" is typically Shift+. on a UK PC keyboard.

Editing C files

This section is for C programmers. There are equivalent sections for other languages. One language will be selected for demonstration in the lecture based on the composition of the audience.

This is dull. All we have done is to edit comments. We will now move on to manipulating the C code itself and see how Emacs' C mode can help us.

Near the bottom of the file is a commented out `fprintf()` statement left over from the writing of the program:



```

if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
/* fprintf(stderr, "%l\t%l\t%l\t%f\n", nxl, xyl, n_iter, epsilon); */
for (ix = 0L; ix < nxl; ix += 1L)
{
}

```

--:-- iterator1.c (C Abbrev)--L92--63%-----

We are going to change the commented line to print the line if a variable “debug” is set to a non-zero value:

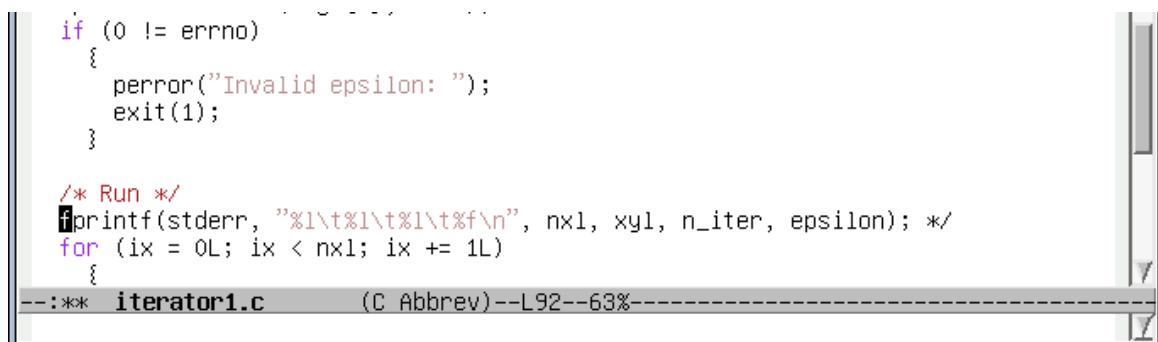
```

if (0 != debug)
{
    fprintf(stderr, "%l\t%l\t%l\t%f\n", nxl, nyl, n_iter, epsilon);
}

```

Removing the comment

We will start by removing the leading “/*”. We move the cursor to the “f” of “fprintf” and then press Backspace three times.



```

if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

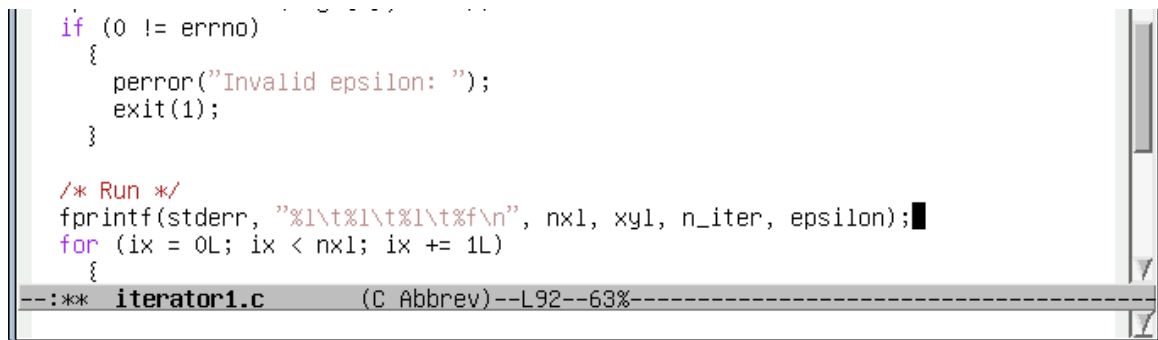
/* Run */
fprintf(stderr, "%l\t%l\t%l\t%f\n", nxl, xyl, n_iter, epsilon); */
for (ix = 0L; ix < nxl; ix += 1L)
{
}

```

--:** iterator1.c (C Abbrev)--L92--63%-----

Observe how the colour-coding changes automatically as soon as the line stops being a comment.

Next we move the cursor to the end of the line (C-e or End) and Backspace over the closing "*/":

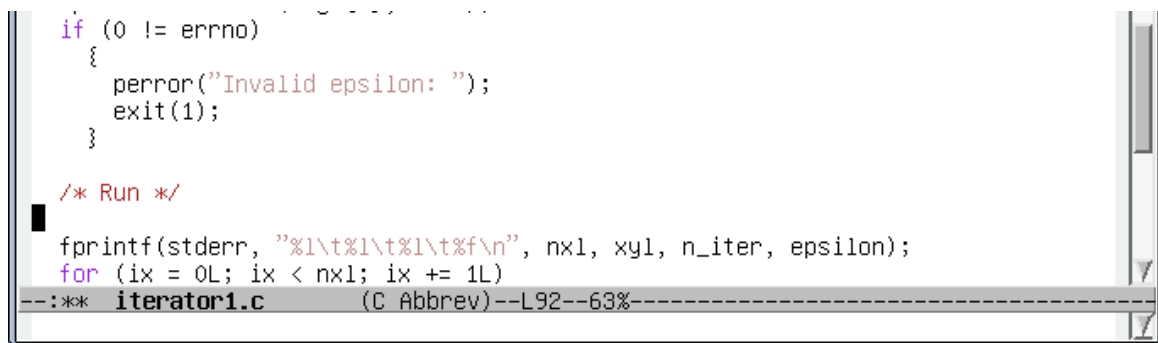


```
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
fprintf(stderr, "%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon);
for (ix = 0L; ix < nx1; ix += 1L)
{
--:*** iterator1.c      (C Abbrev)--L92--63%-----
```

Adding a line above

The easiest way to add a blank line above the fprintf() line (for the if statement) is to move the cursor up one line by pressing the Up arrow key which brings the cursor to the end of the "/* Run */" comment and then to press Return.

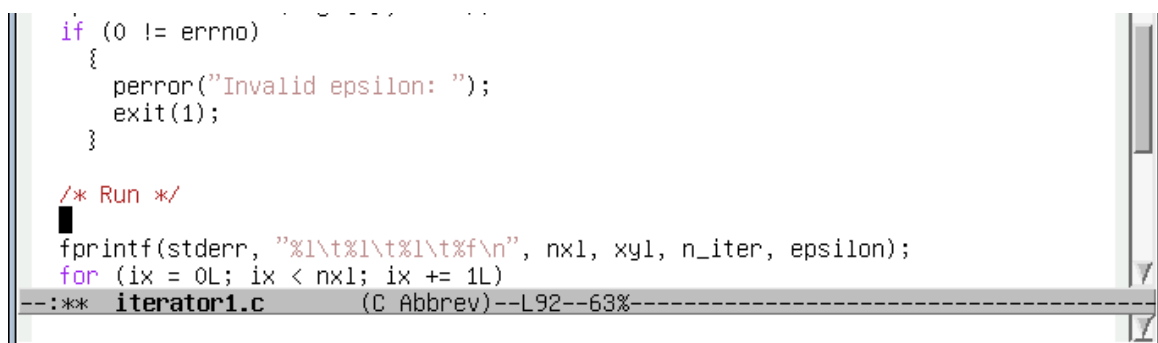


```
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
fprintf(stderr, "%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon);
for (ix = 0L; ix < nx1; ix += 1L)
{
--:*** iterator1.c      (C Abbrev)--L92--63%-----
```

Adding the if statement

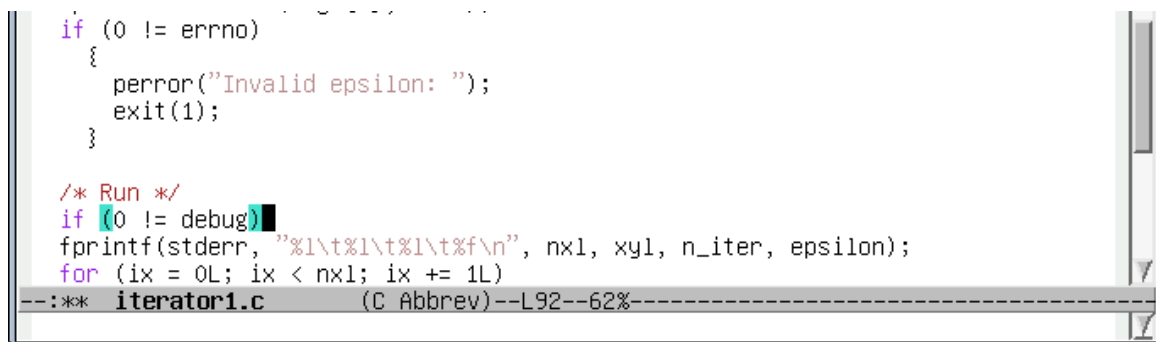
We can put the cursor in the correct indentation by pressing the Tab key.



```
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
fprintf(stderr, "%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon);
for (ix = 0L; ix < nx1; ix += 1L)
{
--:*** iterator1.c      (C Abbrev)--L92--63%-----
```

We can then type the C statement we want. Notice how the word “if” is coloured as Emacs recognizes its status as a keyword:

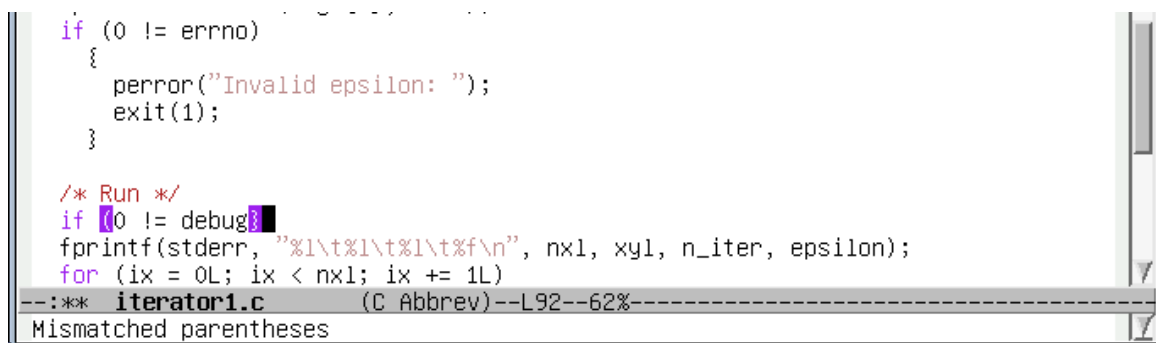


```
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if (0 != debug)
    fprintf(stderr, "%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon);
for (ix = 0L; ix < nx1; ix += 1L)
--:** iterator1.c (C Abbrev)--L92--62%-----
```

Also notice that while the cursor is directly after a closing parenthesis, “)”, both that bracket and the matching open bracket are highlighted in light blue. Only the bracket pair involved with the cursor are highlighted.

If the wrong type of bracket such as a brace, “{”, had been typed then the two brackets that cannot be matched are highlighted in purple:

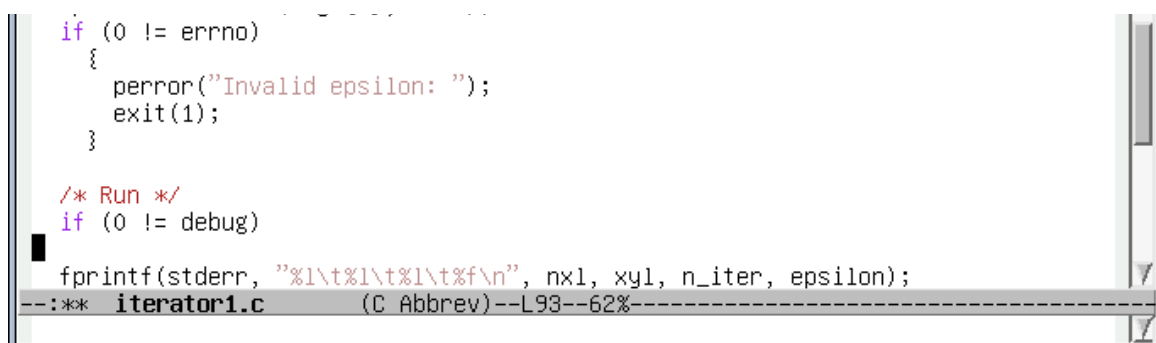


```
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if {0 != debug
    fprintf(stderr, "%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon);
for (ix = 0L; ix < nx1; ix += 1L)
--:** iterator1.c (C Abbrev)--L92--62%-----
Mismatched parentheses
```

Adding the opening brace

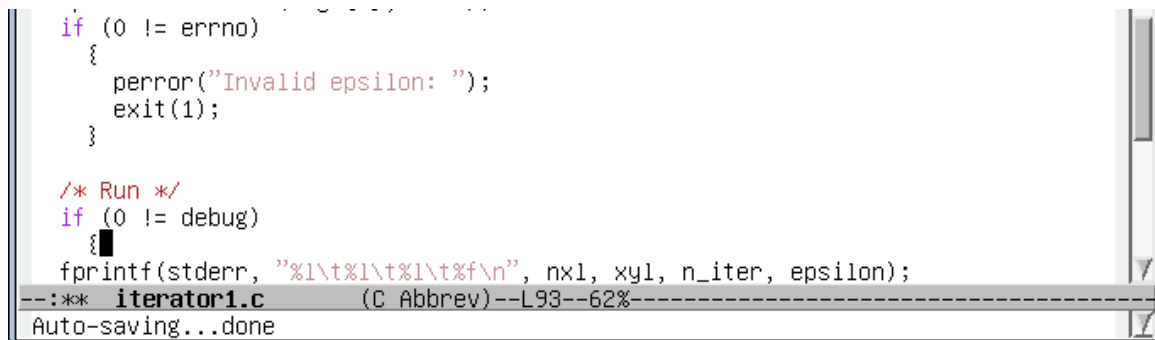
If we hit the Return key at the end of the current line, we start a new line below the if statement, ready for the opening brace, “{”:



```
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if (0 != debug)
{
    fprintf(stderr, "%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon);
--:** iterator1.c (C Abbrev)--L93--62%-----
```


Note how the cursor is at the start of the line. The moment we type "{", though, the cursor jumps across to indent it correctly. (Though the `fprintf()` line below still requires work. We will address this in a moment.)



```

if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if (0 != debug)
{
    fprintf(stderr, "%1\\t%1\\t%1\\t%f\\n", nx1, xy1, n_iter, epsilon);
}

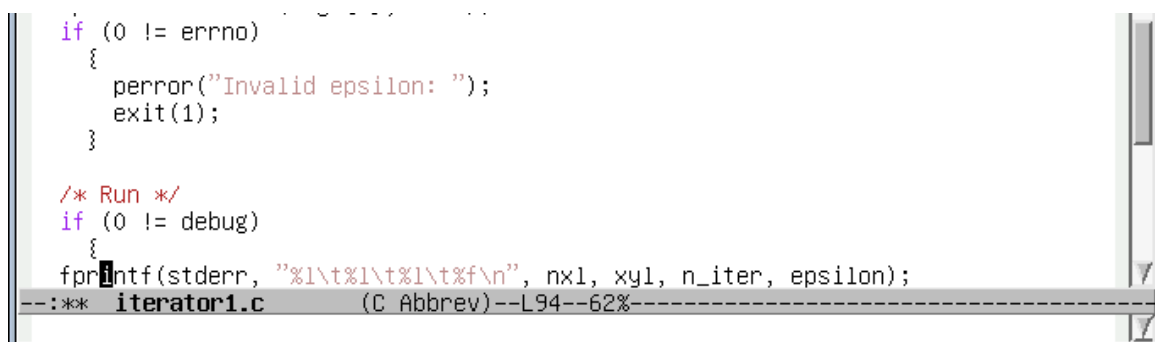
```

--:** iterator1.c (C Abbrev) --L93--62%-----
Auto-saving...done

Indentation and bracket layout in C is a matter of personal choice and group policy. I would advise against fighting Emacs' styles. They are self-consistent and they relieve you of one more thing to worry about. Focus on the content of the program and leave the layout to Emacs.

Indenting the `fprintf()` line

Now we move the cursor down⁸ into the `fprintf()` line. It does not matter where in the line we arrive:



```

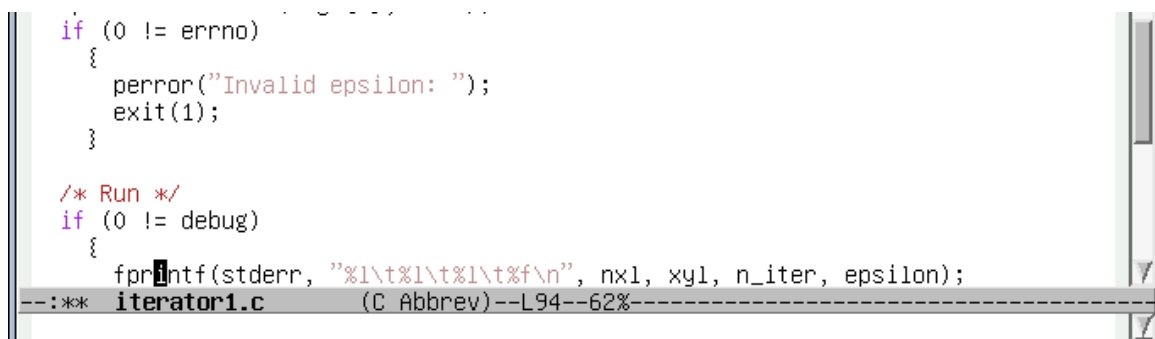
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if (0 != debug)
{
    fprintf(stderr, "%1\\t%1\\t%1\\t%f\\n", nx1, xy1, n_iter, epsilon);
}

```

--:** iterator1.c (C Abbrev) --L94--62%-----

Next, we press Tab. The line indents itself with the cursor maintaining its position relative to the text on the line:



```

if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if (0 != debug)
{
    fprintf(stderr, "%1\\t%1\\t%1\\t%f\\n", nx1, xy1, n_iter, epsilon);
}

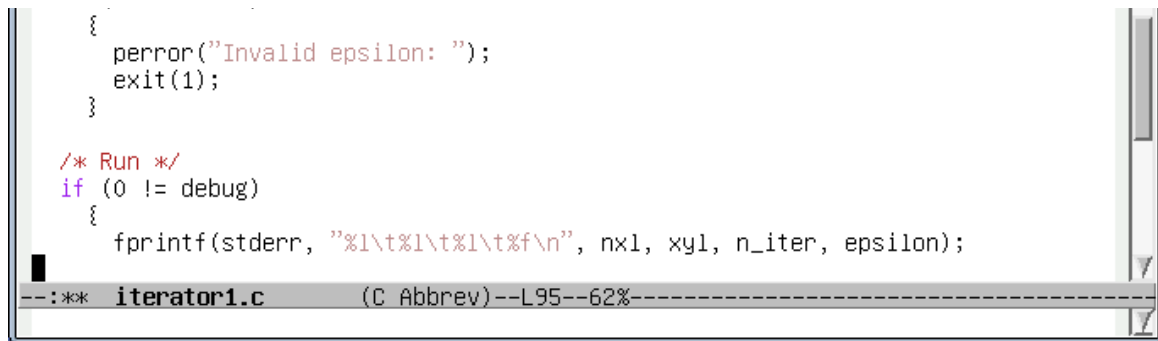
```

--:** iterator1.c (C Abbrev) --L94--62%-----

⁸ Down arrow or C-n.

Adding the closing brace

We need to add the “}” on a line below. We move the cursor to the end of the line⁹ and press Return.



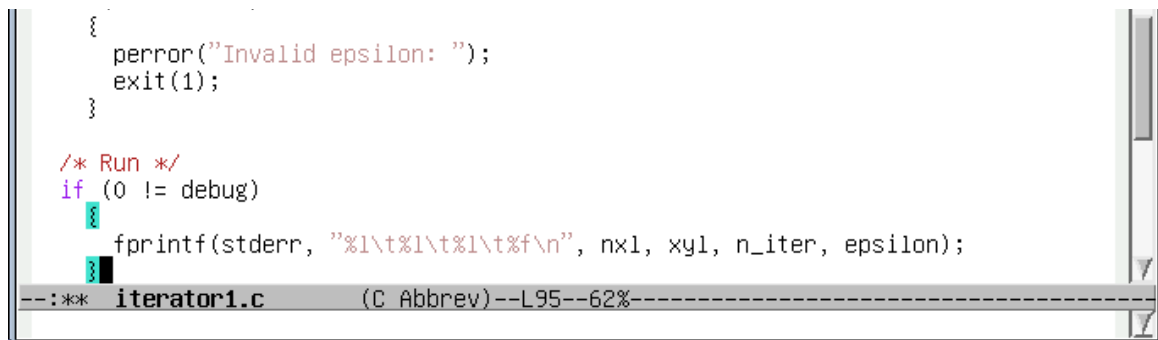
```

{
    perror("Invalid epsilon: ");
    exit(1);
}

/* Run */
if (0 != debug)
{
    fprintf(stderr, "%1\\t%1\\t%1\\t%f\\n", nx1, xy1, n_iter, epsilon);

```

Then we press “}”. The line automatically indents to match the opening brace and highlights the matching pair.



```

{
    perror("Invalid epsilon: ");
    exit(1);
}

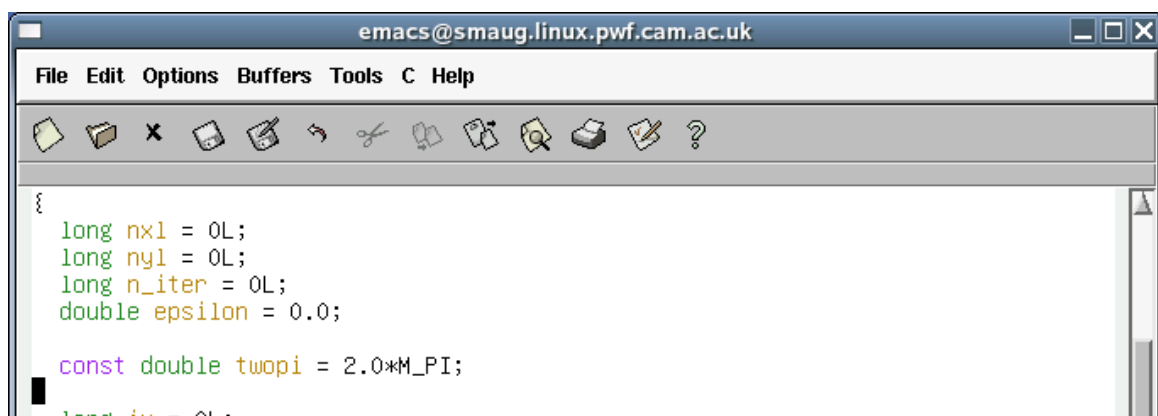
/* Run */
if (0 != debug)
{
    fprintf(stderr, "%1\\t%1\\t%1\\t%f\\n", nx1, xy1, n_iter, epsilon);
}

```

Setting up the debug variable

Finally we have to declare and define the debug variable at the head of the program.

We start by moving to the top of the file¹⁰ and then moving down to the declaration section¹¹:



```

{
    long nx1 = 0L;
    long ny1 = 0L;
    long n_iter = 0L;
    double epsilon = 0.0;

    const double twopi = 2.0*M_PI;

    long ix = 0L;

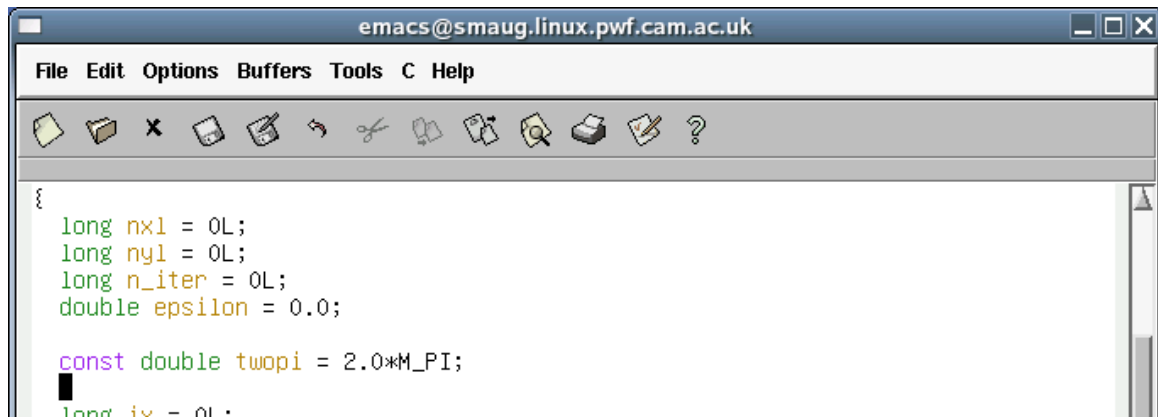
```

⁹ C-e or End.

¹⁰ M-<

¹¹ With a combination of PgDown and Down arrow, typically.

Then we press Tab for the appropriate indentation, though we could do this equally well after we had typed the line.



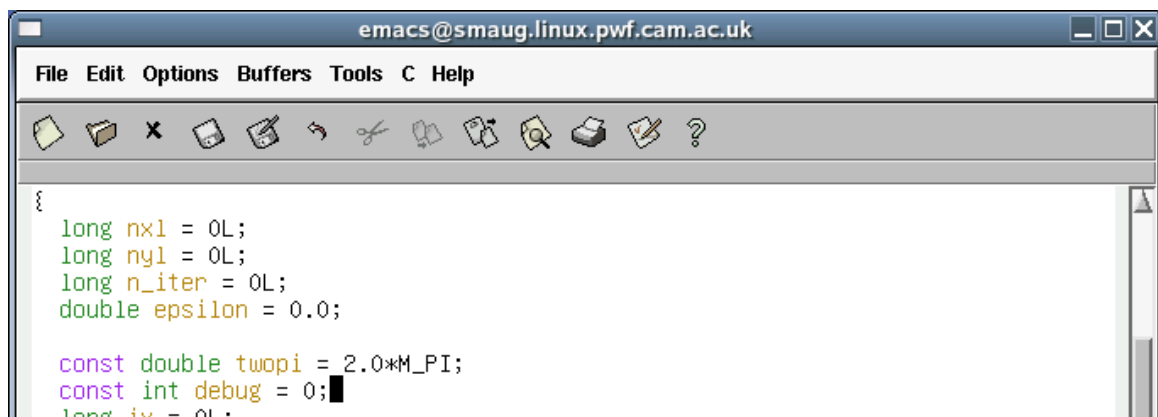
The screenshot shows the Emacs editor window titled 'emacs@smaug.linux.pwf.cam.ac.uk'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'C', and 'Help'. The toolbar contains various icons for file operations and editing. The code in the buffer is as follows:

```
{
  long nxl = 0L;
  long nyl = 0L;
  long n_iter = 0L;
  double epsilon = 0.0;

  const double twopi = 2.0*M_PI;
  long iv = 0L;
```

The cursor is positioned at the start of the line following 'long iv = 0L;'.

Next we type the line of C code required.



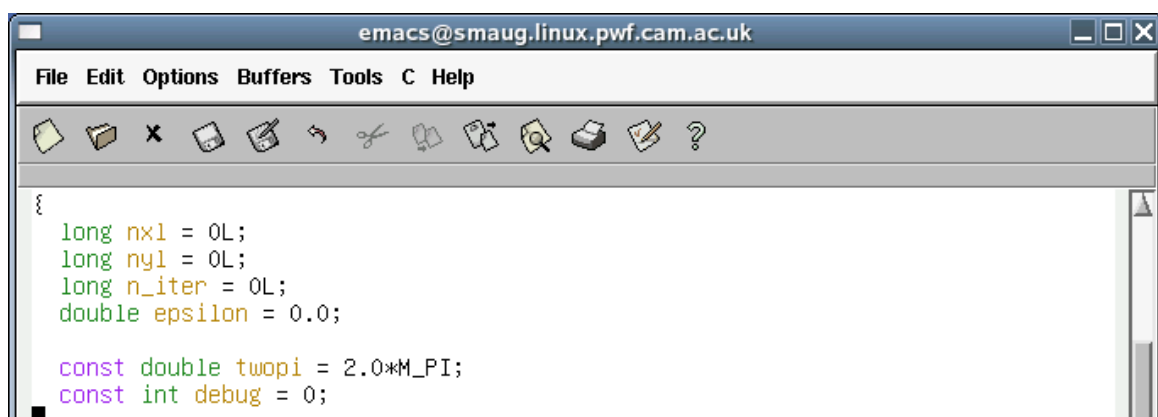
The screenshot shows the Emacs editor window with the same code as before, but with an additional line of code added:

```
{
  long nxl = 0L;
  long nyl = 0L;
  long n_iter = 0L;
  double epsilon = 0.0;

  const double twopi = 2.0*M_PI;
  const int debug = 0;
  long iv = 0L;
```

The cursor is now at the end of the line 'const int debug = 0;'.

and finish off by pressing Return for a fresh blank line:



The screenshot shows the Emacs editor window with the code from the previous screenshot, plus a new blank line at the end:

```
{
  long nxl = 0L;
  long nyl = 0L;
  long n_iter = 0L;
  double epsilon = 0.0;

  const double twopi = 2.0*M_PI;
  const int debug = 0;
  long iv = 0L;

```

The cursor is at the start of the new blank line.

Don't forget to press C-x C-s to save the file before trying to compile it.

```
long count = 0L;
double junk = 0.0; /* Used to discard integer part in modf() */

/* Process the user's command line arguments. */
if (argc != 5)
{
    fprintf(stderr, "Wrong number of arguments!\n4 expected, %d found.\nUsage: %s Nx Ny n_iterations epsilon\n", (argc-1), argv[0]);
    exit(1);
}

--:-- iterator1.c (C Abbrev)--L32--20%-----
Wrote /home/rjd4/EmacsCourse/iterator1.c
```

Editing Fortran files

This section is for Python programmers. There are equivalent sections for other languages. One language will be selected for demonstration in the lecture based on the composition of the audience.

Just as Emacs has a C mode for C programmers, it has a Fortran mode for Fortran programmers. If we load the file `iterator2.f` we see it colour coded according to Fortran's syntax rules and the status line at the bottom declares Emacs to be in Fortran mode.

```

PROGRAM ITER
INTEGER NX
INTEGER NY
INTEGER ITERATIONS
DOUBLE PRECISION EPSILON
DOUBLE PRECISION TWOPI

INTEGER IX
INTEGER IY
DOUBLE PRECISION X
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON
C  WRITE(7,*) NX, NY, ITERATIONS, EPSILON

DO 102 IX = 0, NX
DO 102 IY = 0, NY

    X = IX
  
```

--:-- **iterator2.f** (Fortran)--L1--Top-----

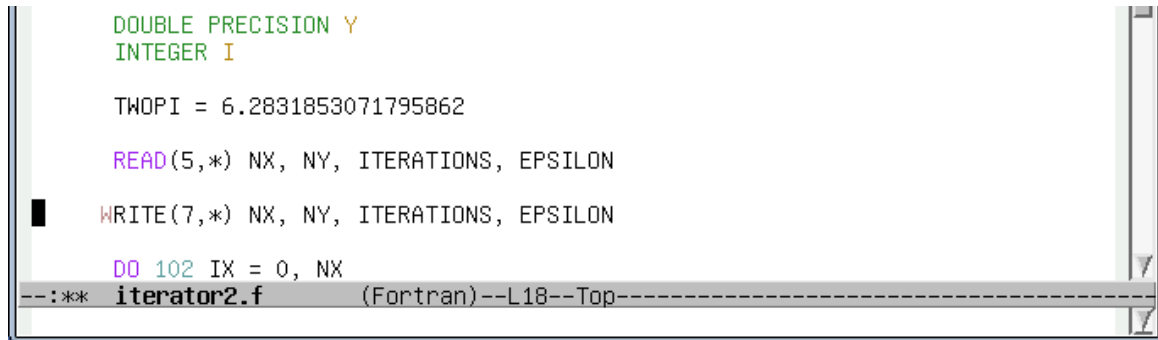
Near the bottom of that screen shot is a commented out `WRITE` statement left over from the writing of the program. We are going to change the commented line to write the data only if a variable `DEBUG` is set to a non-zero value:

```

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
ENDIF
  
```

Removing the comment

Making the line “active” is very easy. We move the cursor to the point just to the right of the “C” in the leading column (i.e. we put the cursor in the second column) and then we press Backspace once. Notice how the line changes colour once it is no longer a comment but instead the line appears in black except for the leading “W” of “WRITE” which highlights the fact that it has slipped into a non-coding column and is acting as a line continuation marker.



```

DOUBLE PRECISION Y
INTEGER I

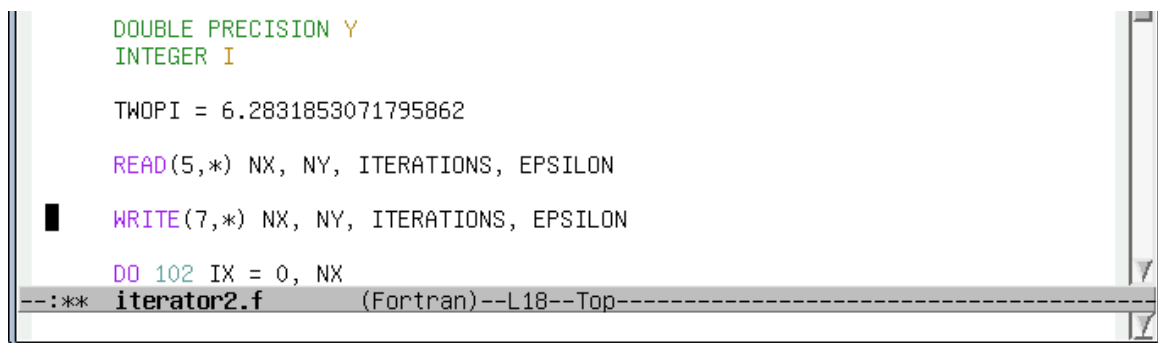
TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON
WRITE(7,*) NX, NY, ITERATIONS, EPSILON

DO 102 IX = 0, NX
--:** iterator2.f (Fortran)--L18--Top-----

```

If we press Space at this point then we re-align the statement to fix this and the colour coding changes again to indicate the status of the various elements in Fortran:



```

DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

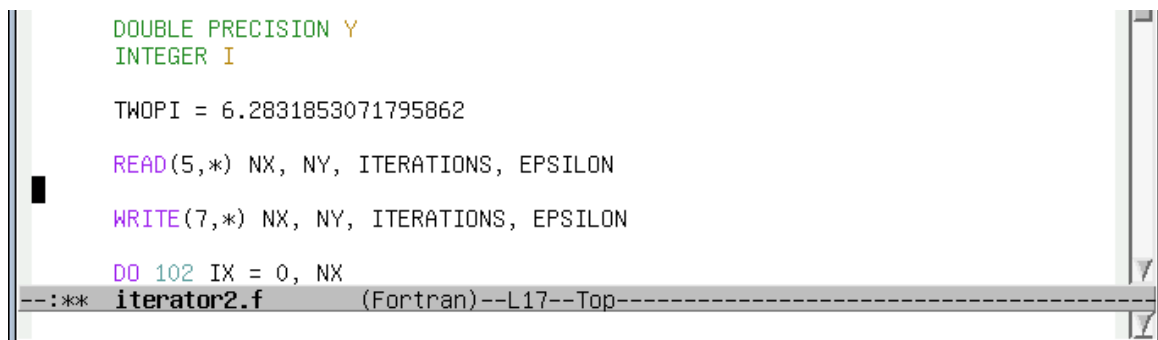
READ(5,*) NX, NY, ITERATIONS, EPSILON
WRITE(7,*) NX, NY, ITERATIONS, EPSILON

DO 102 IX = 0, NX
--:** iterator2.f (Fortran)--L18--Top-----

```

Adding a line above

The easiest way to add a blank line above the WRITE statement (for the IF statement to occupy) is to move the cursor up one line by pressing the Up arrow which brings the cursor to the start of the blank line between the READ and the WRITE.



```

DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

WRITE(7,*) NX, NY, ITERATIONS, EPSILON

DO 102 IX = 0, NX
--:** iterator2.f (Fortran)--L17--Top-----

```

Now we just press Return:

```
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

WRITE(7,*) NX, NY, ITERATIONS, EPSILON

--:** iterator2.f      (Fortran)--L18--Top-----
```

Adding the IF statement

We can put the cursor in the correct indentation by pressing the Tab key.

```
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

WRITE(7,*) NX, NY, ITERATIONS, EPSILON

--:** iterator2.f      (Fortran)--L18--Top-----
```

We can then type the Fortran statement we want. Notice how the word “IF” is coloured as Emacs recognizes its status as a keyword:

```
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

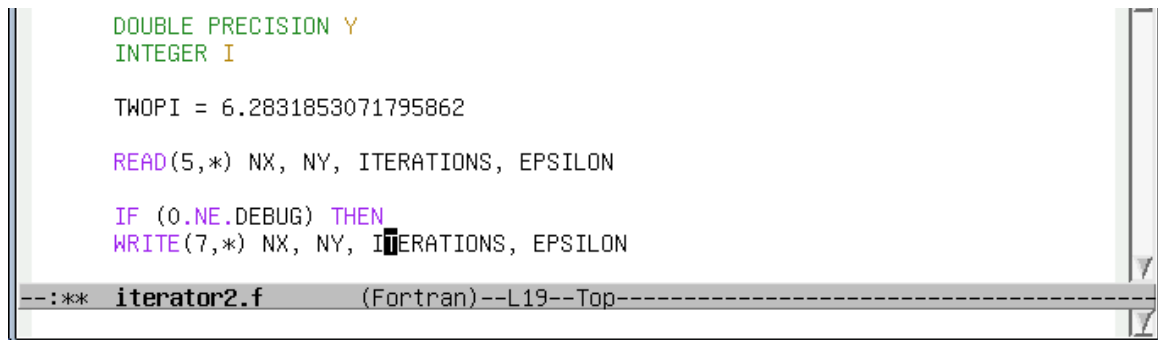
READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
WRITE(7,*) NX, NY, ITERATIONS, EPSILON

--:** iterator2.f      (Fortran)--L18--Top-----
```

Indenting the WRITE line

Now we move the cursor down¹² to the WRITE line below. It does not matter where in the line we arrive:



```

DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

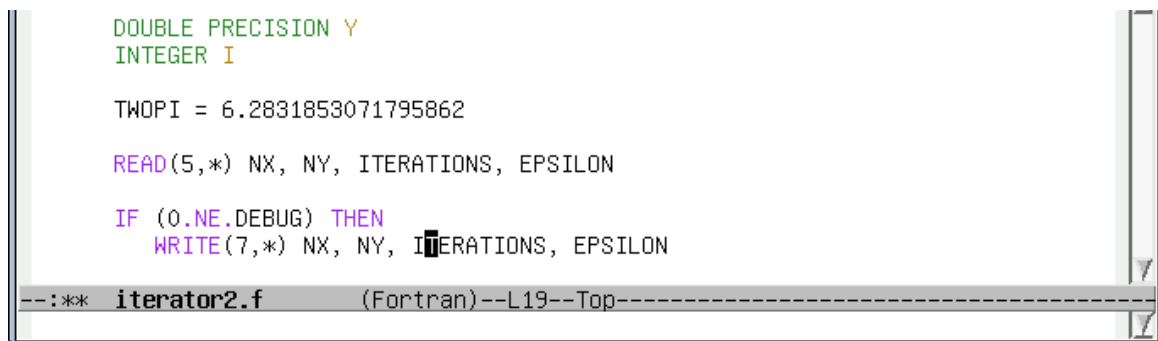
READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
WRITE(7,*) NX, NY, ITERATIONS, EPSILON

```

--:** iterator2.f (Fortran)--L19--Top--

Next, we press Tab. Again this triggers alignment. The line indents itself with the cursor maintaining its position relative to the text on the line:



```

DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

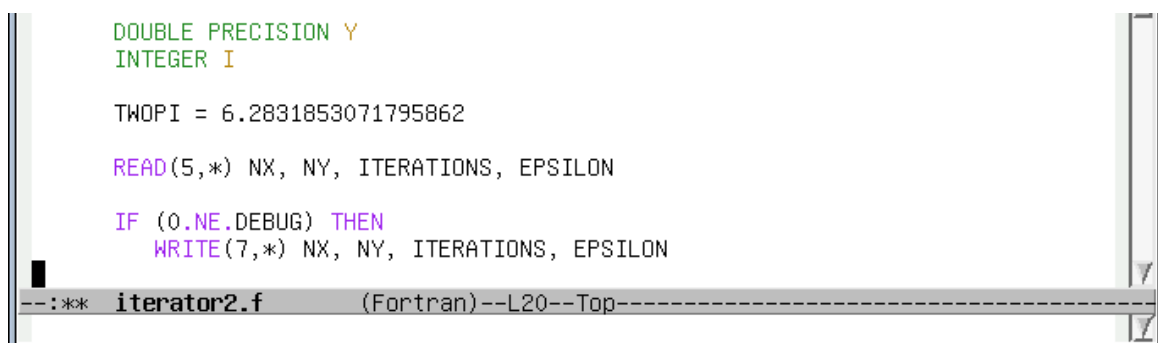
IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON

```

--:** iterator2.f (Fortran)--L19--Top--

Adding the ENDIF statement

We need to add the ENDIF on the line below. we move the cursor to the end of the current line¹³ and press Return:



```

DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON

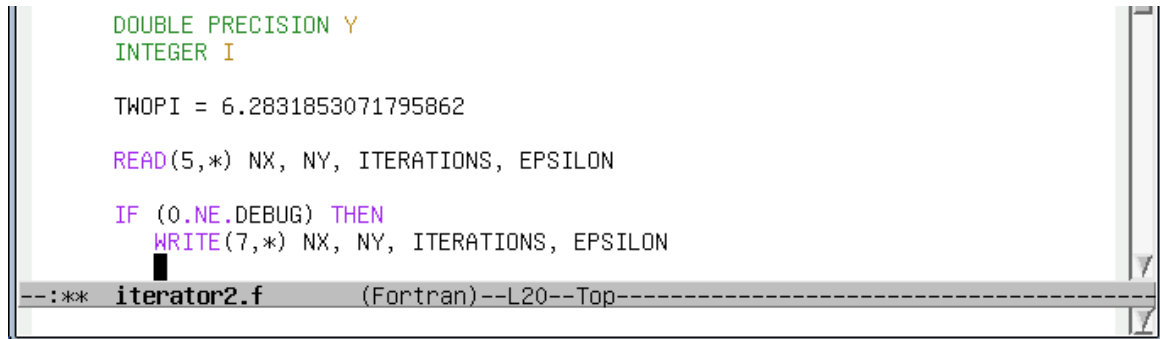
```

--:** iterator2.f (Fortran)--L20--Top--

¹² Down arrow or C-n.

¹³ End or C-e.

We press Tab to move the cursor into position. Note that the cursor is aligned under the “W” of WRITE ready for a continuation of the block of code that will be executed only if DEBUG is non-zero (the “THEN block”)



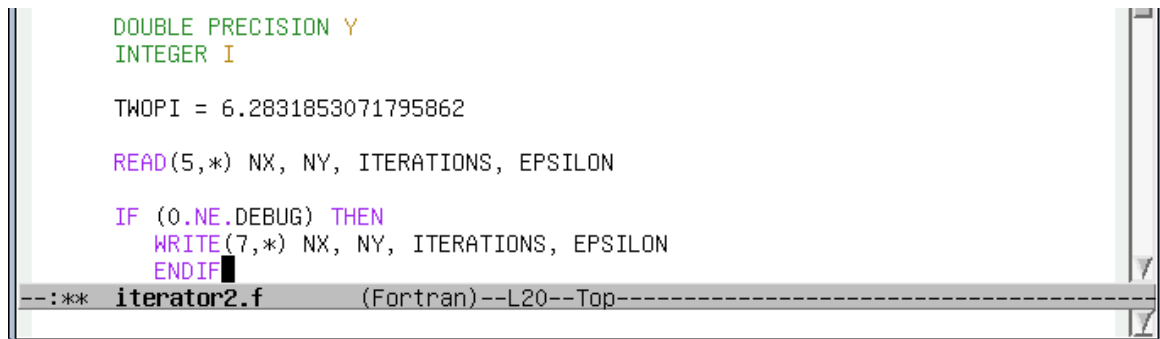
```
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
--:** iterator2.f (Fortran)--L20--Top-----
```

Next we type the ENDIF statement. Emacs colours it for us.



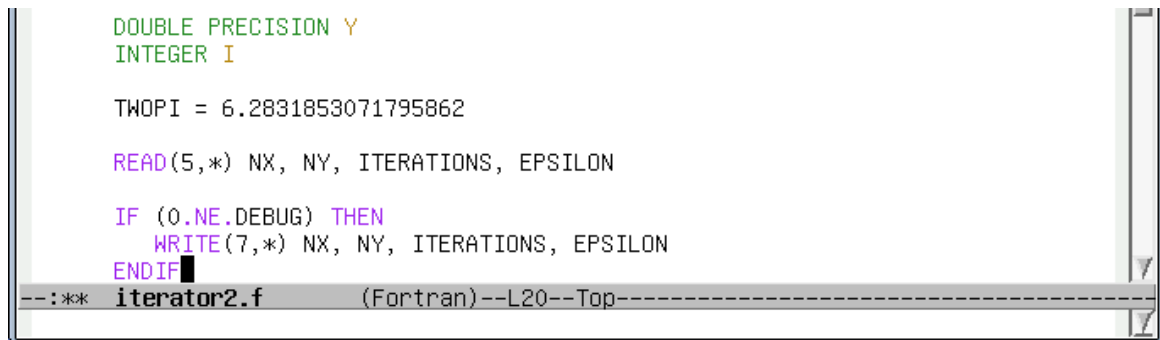
```
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
ENDIF
--:** iterator2.f (Fortran)--L20--Top-----
```

If we press Tab again Emacs now has enough information to know that this is the close of the THEN block and the indentation is corrected:



```
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

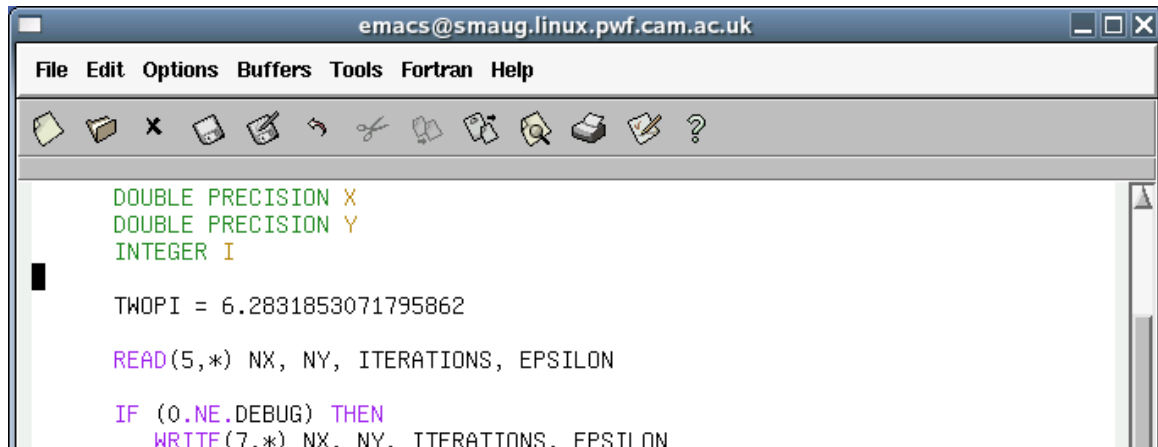
READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
ENDIF
--:** iterator2.f (Fortran)--L20--Top-----
```

Setting up the DEBUG variable

Finally we have to declare and define the DEBUG variable at the dead of the program.

We start by moving to the top of the file¹⁴ and then moving to the end of the declarations:



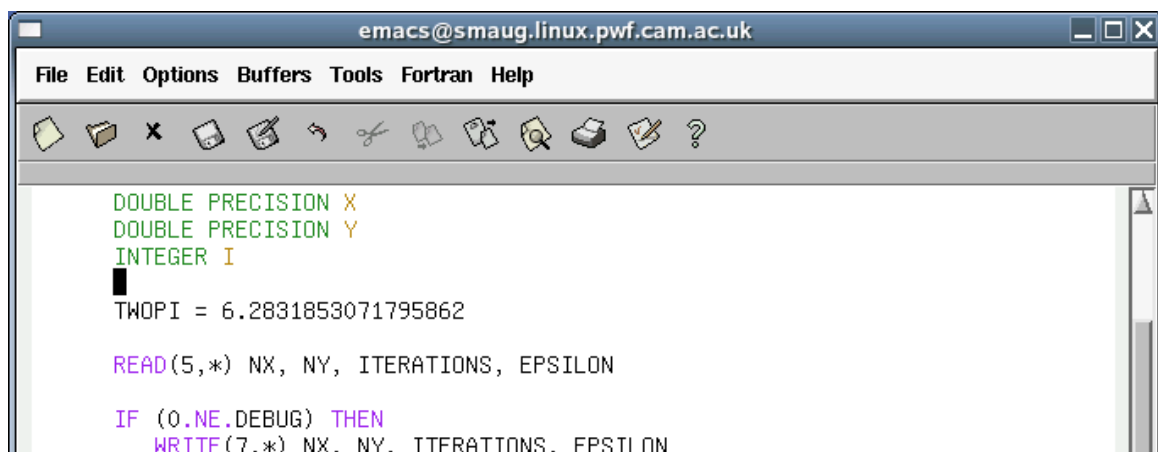
```
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools Fortran Help
DOUBLE PRECISION X
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
```

Next we press Tab for the appropriate indentation:



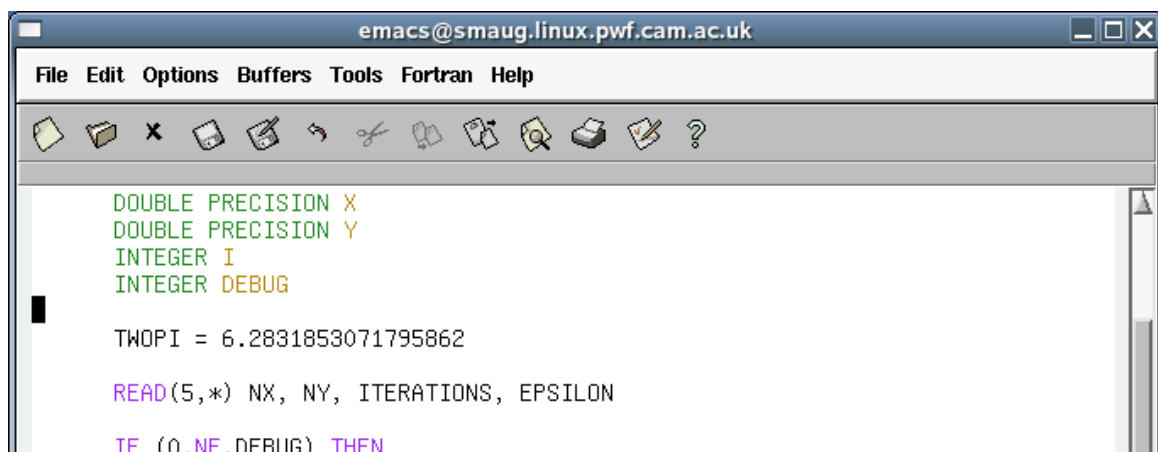
```
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools Fortran Help
DOUBLE PRECISION X
DOUBLE PRECISION Y
INTEGER I

TWOPI = 6.2831853071795862

READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
```

and enter the declaration followed by Return:



```
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools Fortran Help
DOUBLE PRECISION X
DOUBLE PRECISION Y
INTEGER I
INTEGER DEBUG

TWOPI = 6.2831853071795862

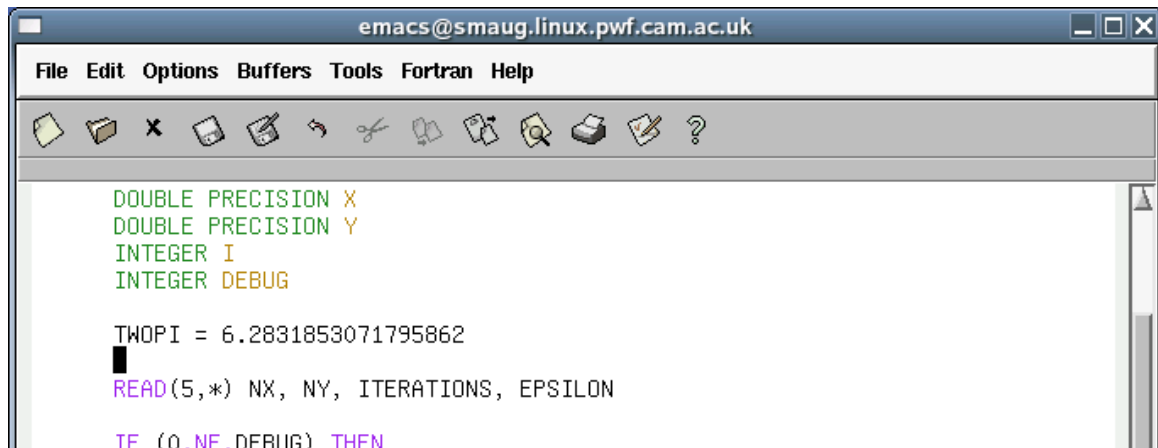
READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
```

¹⁴ M-<

Introduction to Emacs

We move to the line after the TWOPI definition to add the definition of DEBUG, pressing Tab again to get the indentation right:

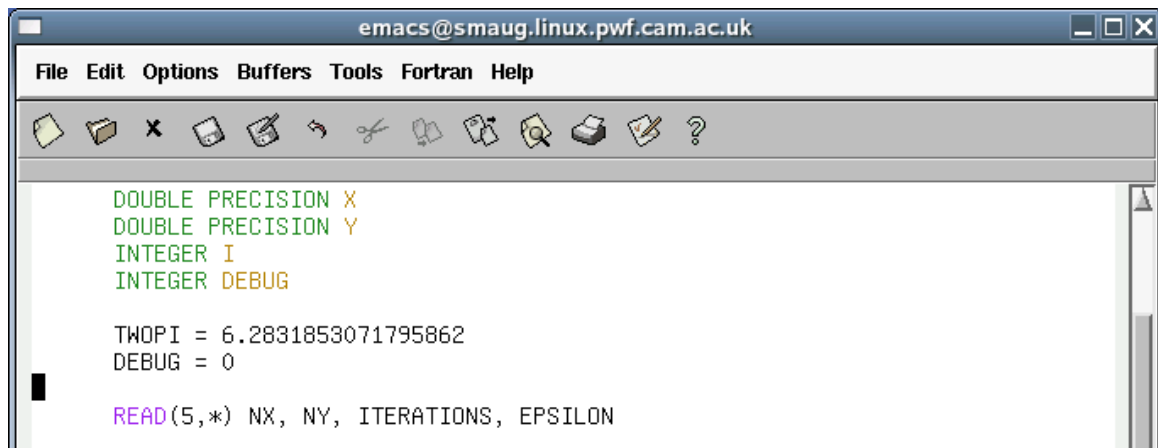


```
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools Fortran Help
DOUBLE PRECISION X
DOUBLE PRECISION Y
INTEGER I
INTEGER DEBUG

TWOPI = 6.2831853071795862
READ(5,*) NX, NY, ITERATIONS, EPSILON

IF (0.NE.DEBUG) THEN
```

and then enter the code itself followed by Return:

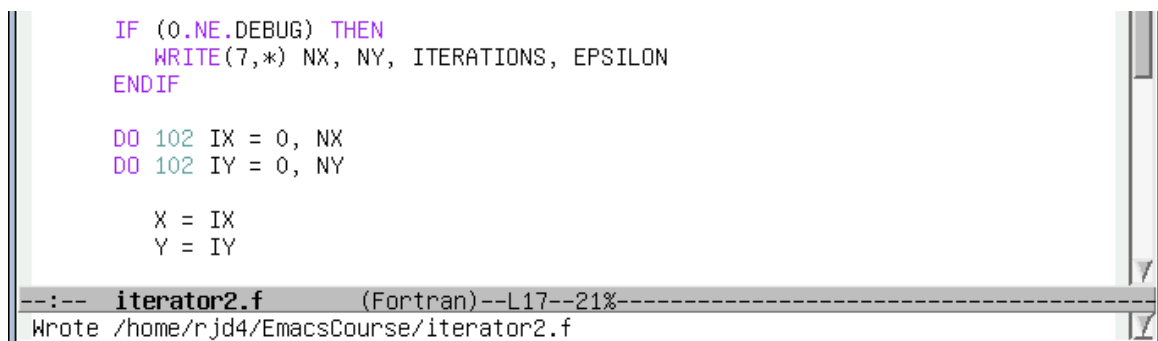


```
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools Fortran Help
DOUBLE PRECISION X
DOUBLE PRECISION Y
INTEGER I
INTEGER DEBUG

TWOPI = 6.2831853071795862
DEBUG = 0

READ(5,*) NX, NY, ITERATIONS, EPSILON
```

Don't forget to press C-x C-s to save the file before trying to compile it.



```
IF (0.NE.DEBUG) THEN
  WRITE(7,*) NX, NY, ITERATIONS, EPSILON
ENDIF

DO 102 IX = 0, NX
DO 102 IY = 0, NY

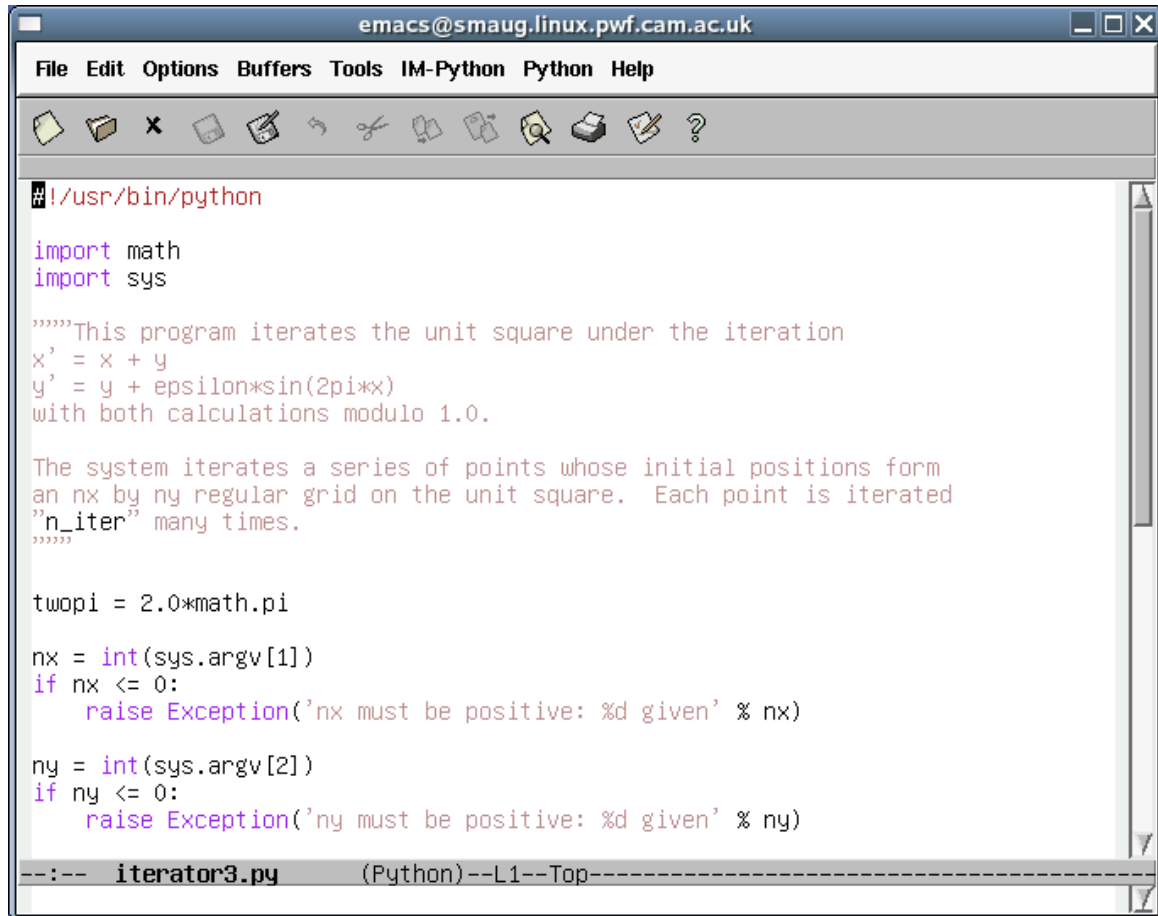
  X = IX
  Y = IY

---:-- iterator2.f (Fortran)--L17--21%-----
Wrote /home/rjd4/EmacsCourse/iterator2.f
```

Editing Python files

This section is for Python programmers. There are equivalent sections for other languages. One language will be selected for demonstration in the lecture based on the composition of the audience.

Just as Emacs has a C mode for C programmers, it has a Python mode for Python programmers. If we load the file `iterator3.py` we see it colour coded according to Python's syntax rules and the status line at the bottom declares Emacs to be in Python mode.



```
#!/usr/bin/python

import math
import sys

"""This program iterates the unit square under the iteration
x' = x + y
y' = y + epsilon*sin(2pi*x)
with both calculations modulo 1.0.

The system iterates a series of points whose initial positions form
an nx by ny regular grid on the unit square. Each point is iterated
""n_iter"" many times.
"""

twopi = 2.0*math.pi
nx = int(sys.argv[1])
if nx <= 0:
    raise Exception('nx must be positive: %d given' % nx)

ny = int(sys.argv[2])
if ny <= 0:
    raise Exception('ny must be positive: %d given' % ny)

--:-- iterator3.py (Python)--L1--Top--
```

Near the bottom of the file is a commented out print statement left over from the writing of the program:

```

emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools IM-Python Python Help

ny = int(sys.argv[2])
if ny <= 0:
    raise Exception('ny must be positive: %d given' % ny)

n_iter = int(sys.argv[3])
if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

epsilon = float(sys.argv[4])
# print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(nx)
        y = float(iy)/float(ny)
        for count in range(0, n_iter):
            xx = x + y
            yy = y*(1.0 + epsilon*math.sin(twopi*x))
            x = xx - math.floor(xx)
            y = yy - math.floor(yy)
        print "%f\t%f" % (x,y)

--:-- iterator3.py (Python)--L31--Bot-----

```

We are going to change the commented line to print the line if a variable “debug” is set to True:

```

if debug:
    printf '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)

```

Removing the comment

Making the line “active” is very easy. We move the cursor to the “p” of print and press Backspace twice. Notice how the line changes colour as it stops being a comment and becomes live Python, subject to syntactic analysis.

```

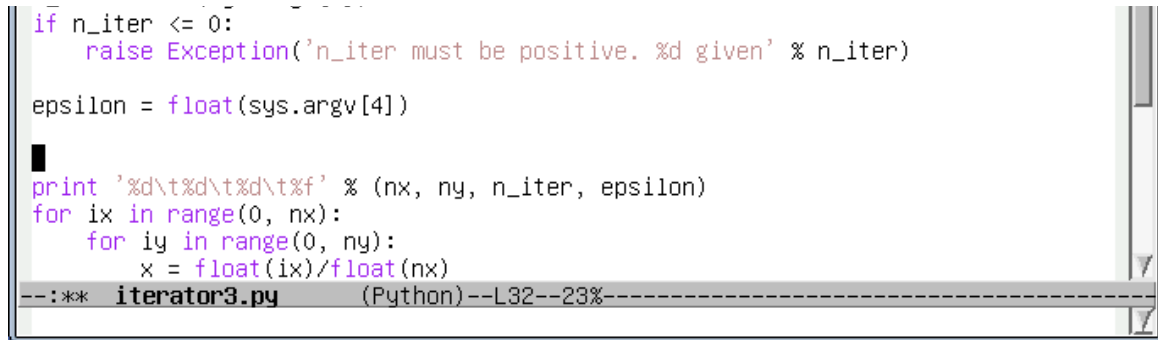
if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

epsilon = float(sys.argv[4])
print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(nx)
        y = float(iy)/float(ny)
--:** iterator3.py (Python)--L32--24%-----

```

Adding a line above

We are going to add the `if` statement above the `print` statement. So we move the cursor up one line into the existing blank line and press `Return` once to give us some extra space.



```

if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

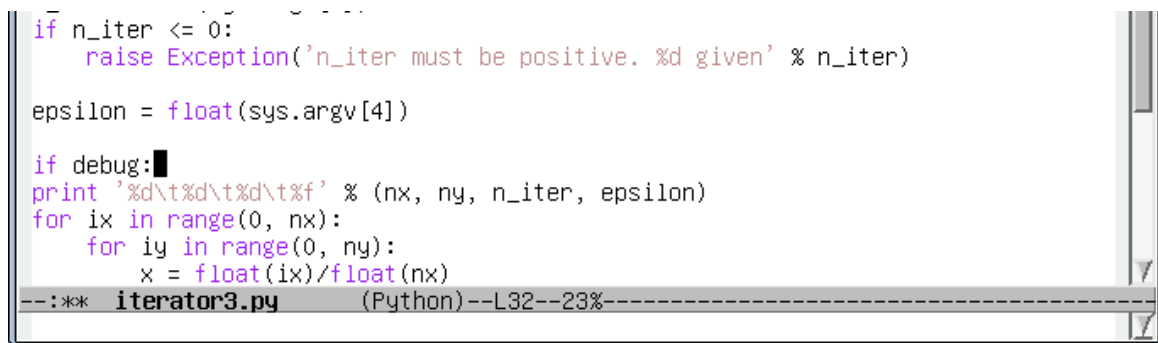
epsilon = float(sys.argv[4])

print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(nx)
--:** iterator3.py (Python)--L32--23%-----

```

Adding the `if` statement

We simply type the relevant Python code. Note how the colours are assigned as we go along.



```

if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

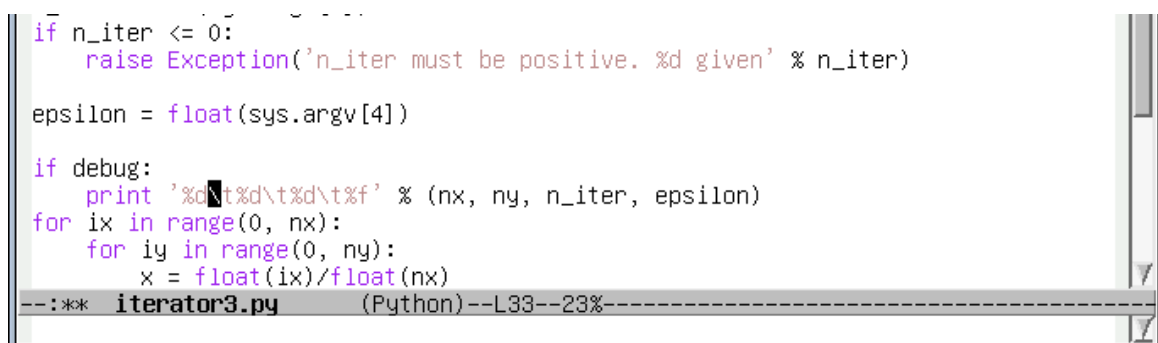
epsilon = float(sys.argv[4])

if debug:
print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(nx)
--:** iterator3.py (Python)--L32--23%-----

```

Indenting the `print` line

If we now move the cursor down into the `print` line below, and anywhere in that line will do, we can press `Tab` to indent the line appropriately.



```

if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

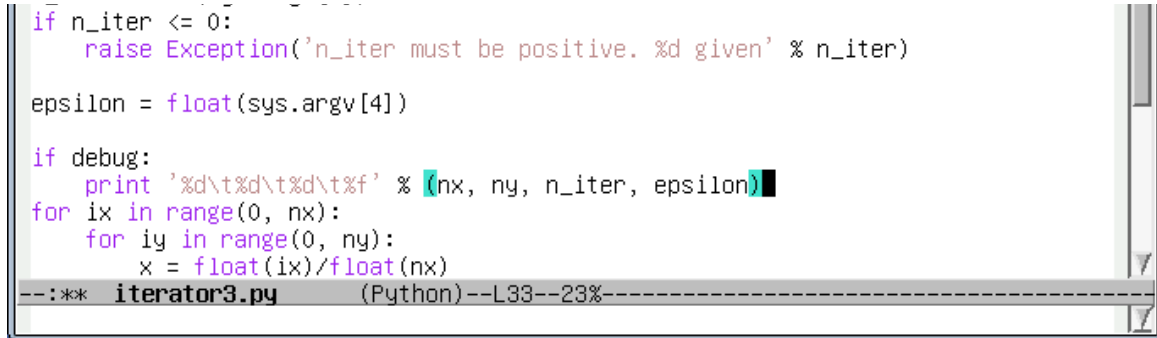
epsilon = float(sys.argv[4])

if debug:
    print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(nx)
--:** iterator3.py (Python)--L33--23%-----

```

Bracket matching

If we want to add a line below the if/print lines we move the cursor to the end of the print line with C-e or End and press Return. Note that while the cursor is at the end of the line, just after the closing bracket of the "(nx, ny, n_iter, epsilon)" tuple both brackets light up in pale blue:



```

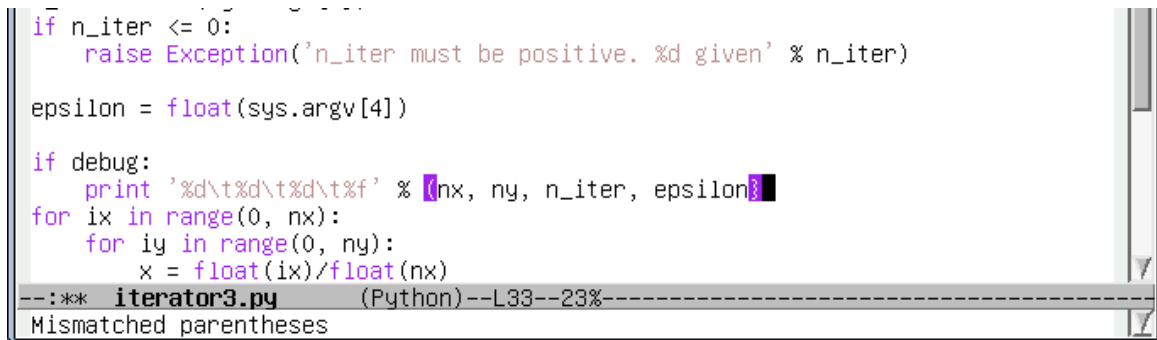
if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

epsilon = float(sys.argv[4])

if debug:
    print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(ny)
--:** iterator3.py (Python)--L33--23%-----

```

Emacs uses this colour coding to help you match brackets correctly. Just for a moment, backspace over the closing parenthesis, ")", and replace it with a closing brace, "}". The matching turns purple to warn you that the brackets don't match correctly and a warning appears in the mini-buffer:



```

if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

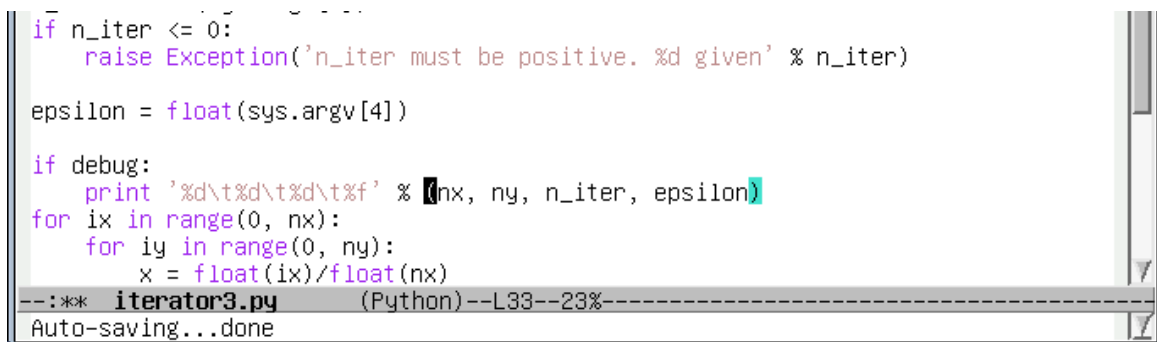
epsilon = float(sys.argv[4])

if debug:
    print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)}
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(ny)
--:** iterator3.py (Python)--L33--23%-----
Mismatched parentheses

```

We'll put back the parenthesis to make the program syntactically correct. This is an Emacs tutorial, not a Python one.

The colour coding is only displayed for the brackets involved with the cursor. To be precise, this is when the cursor follows a closing bracket or is on an opening bracket. If we move the cursor back to the opening bracket we see this:



```

if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

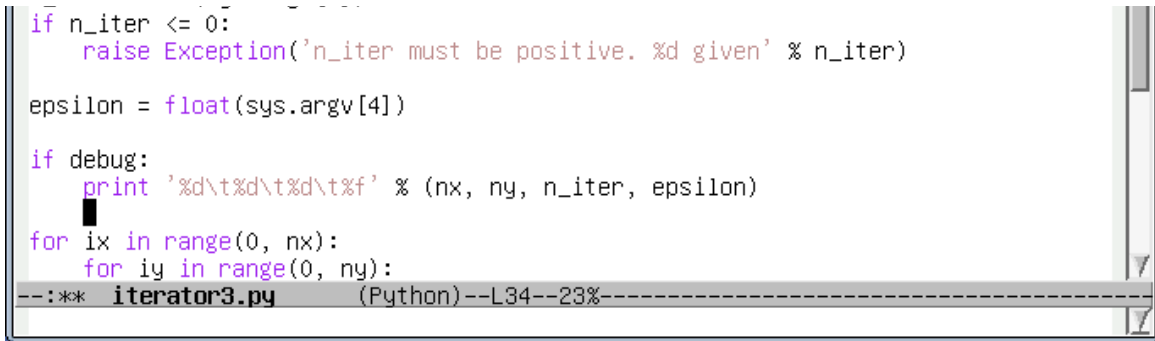
epsilon = float(sys.argv[4])

if debug:
    print '%d\t%d\t%d\t%f' % (nx, ny, n_iter, epsilon)
for ix in range(0, nx):
    for iy in range(0, ny):
        x = float(ix)/float(ny)
--:** iterator3.py (Python)--L33--23%-----
Auto-saving...done

```

Adding a line below

By now you should know what to do. Move to the end of the line (C-e or End) and press Return.



```

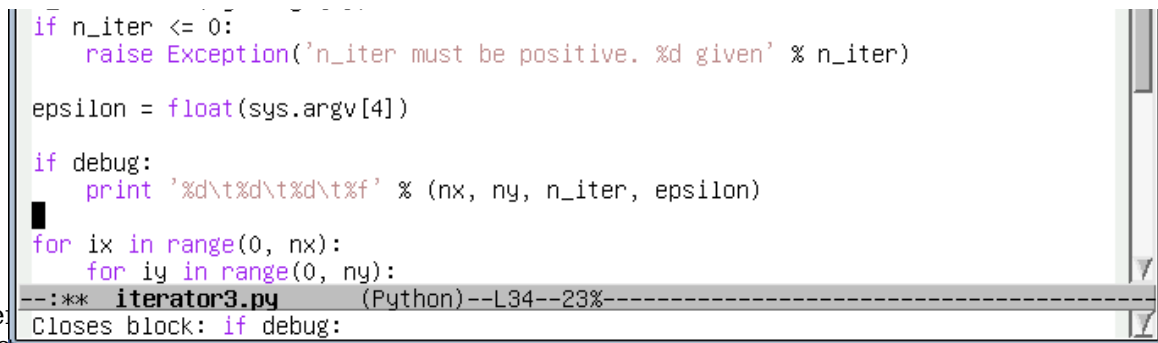
if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

epsilon = float(sys.argv[4])

if debug:
    print '%d\\t%d\\t%d\\t%f' % (nx, ny, n_iter, epsilon)
    █
for ix in range(0, nx):
    for iy in range(0, ny):
--:* iterator3.py (Python)--L34--23%

```

Notice how the cursor moves to a position immediately under the “p” of “print”. Emacs supports the indenting system used by Python and each time you start a new line it makes an educated guess¹⁵ as to where you want the cursor. Typically it gets it right. If you press Backspace the indentation decreases one level, in this case to the start of the line:



```

if n_iter <= 0:
    raise Exception('n_iter must be positive. %d given' % n_iter)

epsilon = float(sys.argv[4])

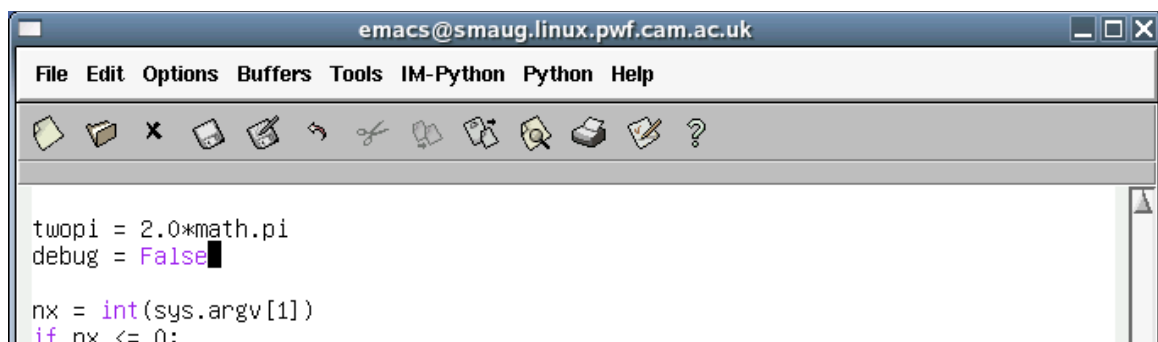
if debug:
    print '%d\\t%d\\t%d\\t%f' % (nx, ny, n_iter, epsilon)
    █
for ix in range(0, nx):
    for iy in range(0, ny):
--:* iterator3.py (Python)--L34--23%

```

When you close a block, the statement that opened the block is reported in the mini-buffer as shown.

Setting up the debug variable

Finally we need to set the debug variable at the top of the file. The M-< command will take the cursor to the very start of the file and we can add the debug definition after twopi.



```

twopi = 2.0*math.pi
debug = False
█

nx = int(sys.argv[1])
if nx <= 0:

```

Note how False is recognised as a Python keyword and displayed in lurid pink.

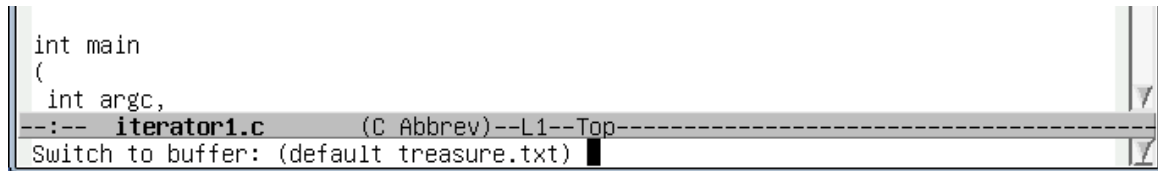
Now that we have made this change to the code don't forget to save the buffer back to disc with C-x C-s.

¹⁵For the Pythonistas: Emacs selects the rightmost position consistent with the preceding code. So it will continue an indented block you are in, indent another level if the preceding line ended with a colon, and only decrease the indentation if the preceding line was a return statement or other block ending command.

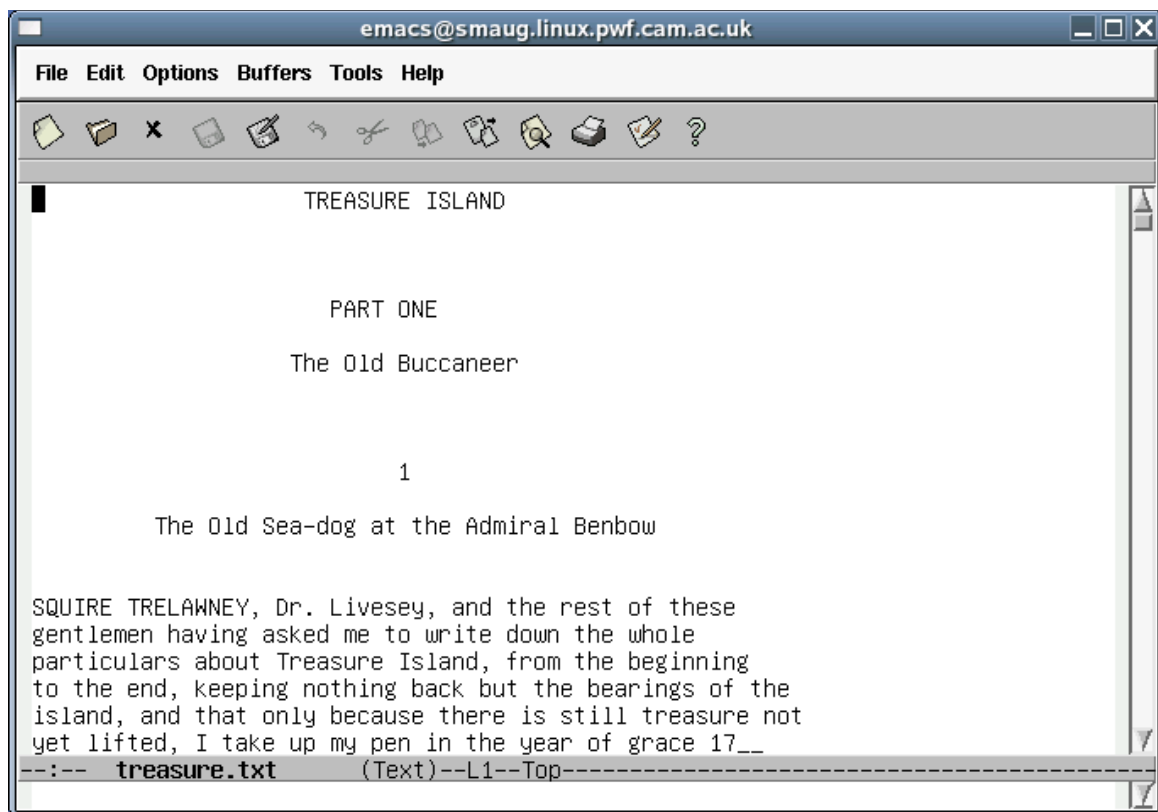
Switching between buffers

We have opened one directory and two files so far. We haven't formally closed any of them. Actually, they are all open. We can switch between them with a variety of options.

The command C-x b¹⁶ will switch the buffer being shown from the current buffer (`iterator1.c`, or whichever program file we used) to the other buffer most recently viewed (`treasure.txt`):



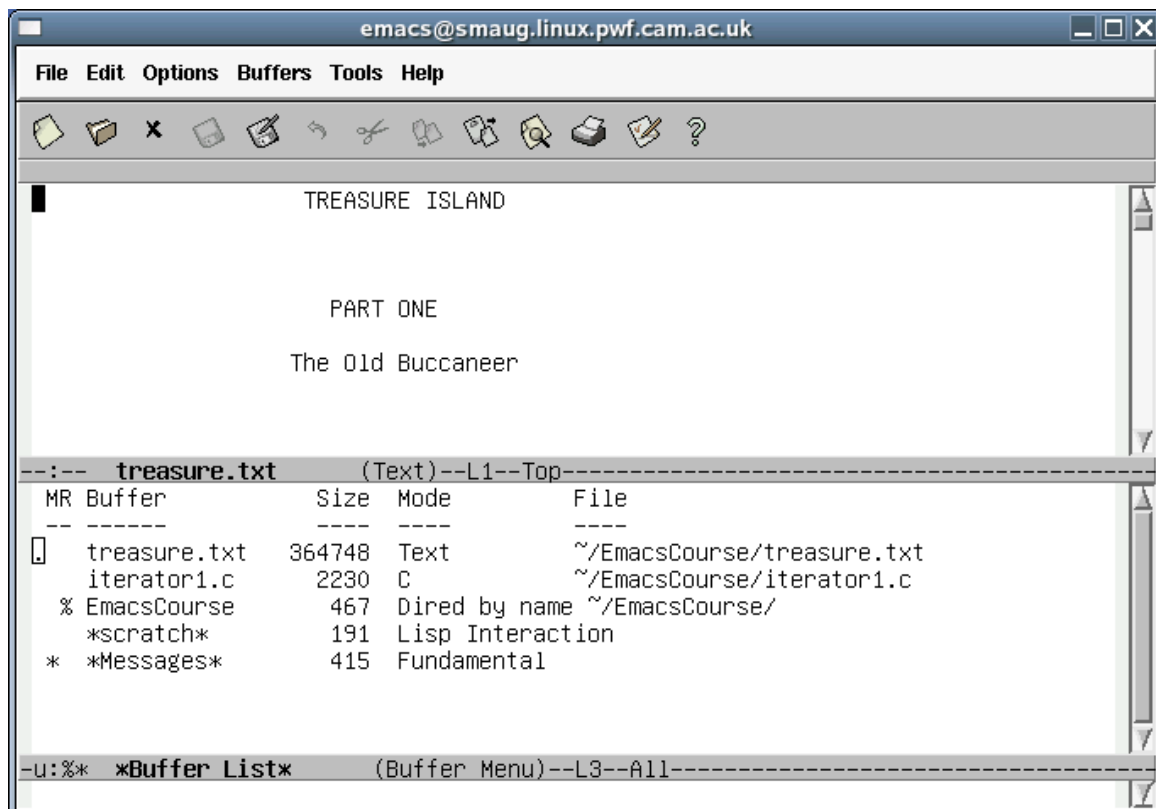
Simply hit Return to confirm the choice:



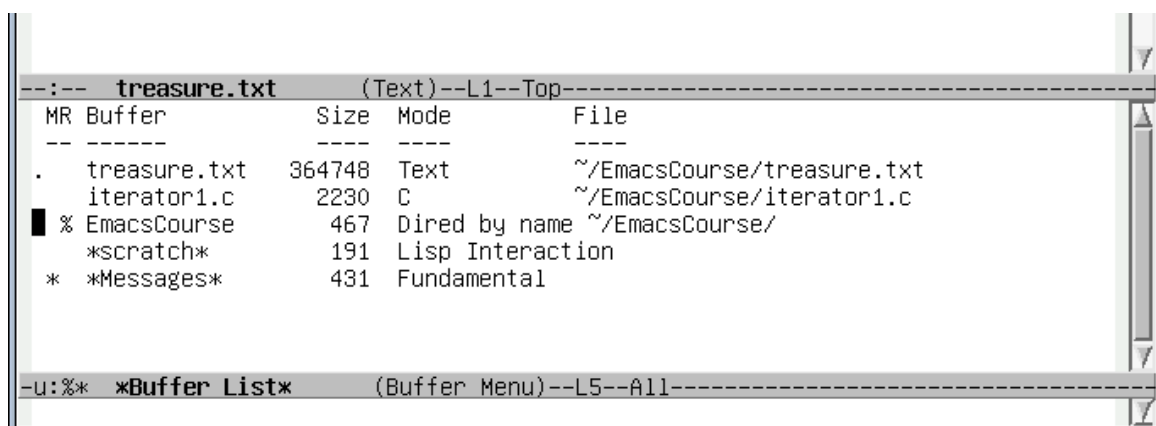
If we run C-x b again we will switch back to `iterator1.c`.

¹⁶ Remember, this means Control+X followed by plain B (i.e. *not* Control+B)

We can get a list of all the currently open buffers by pressing C-x C-b. This has a very dramatic effect visually:

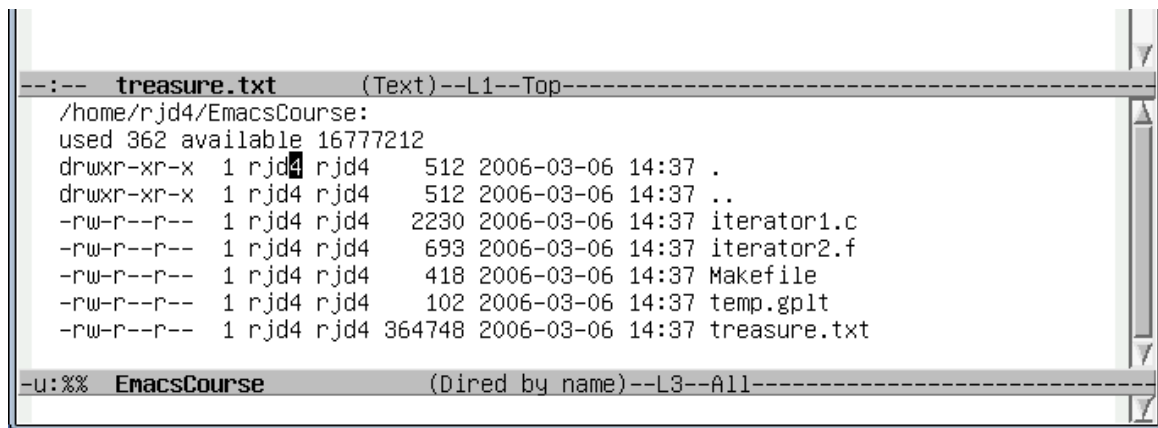


It has split the main window into two parts. The cursor is still in the treasure.txt buffer so to select from one of the offered buffers we need to get the cursor into the other half. There are two ways to do this. We can just click on a buffer name with the mouse to move the cursor there. (The cursor can be moved by the mouse under all circumstances.) Alternatively we can give the instruction C-x o¹⁷ to switch between the halves and once we are in the buffer list we can use the arrow keys to move up and down the list. For example we can move it to the directory listing:



¹⁷That's the letter "oh" rather than the numeral "zero".

Once we are on the right row, we hit Return to select that buffer. It opens in that lower half window:

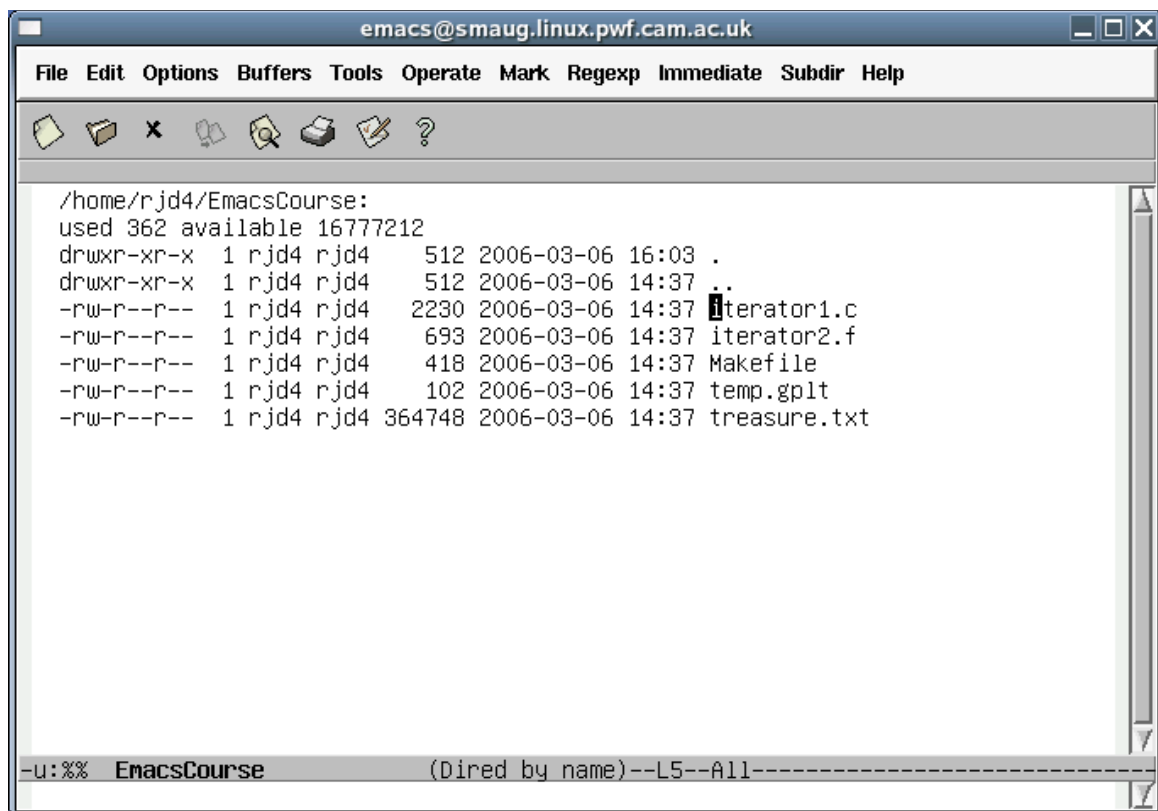


```
--:-- treasure.txt (Text)--L1--Top-----
/home/rjd4/EmacsCourse:
used 362 available 16777212
drwxr-xr-x  1 rjd4 rjd4    512 2006-03-06 14:37 .
drwxr-xr-x  1 rjd4 rjd4    512 2006-03-06 14:37 ..
-rw-r--r--  1 rjd4 rjd4  2230 2006-03-06 14:37 iterator1.c
-rw-r--r--  1 rjd4 rjd4    693 2006-03-06 14:37 iterator2.f
-rw-r--r--  1 rjd4 rjd4    418 2006-03-06 14:37 Makefile
-rw-r--r--  1 rjd4 rjd4    102 2006-03-06 14:37 temp.gplt
-rw-r--r--  1 rjd4 rjd4 364748 2006-03-06 14:37 treasure.txt

-u:%% EmacsCourse (Dired by name)--L3--All-----
```

So now we have both `treasure.txt` and the listing of `EmacsCourse` on display.

If we want just one file on display then move the cursor to the one wanted using `C-x o` to switch between them and then hit `C-x 1` to return to single window mode:



```
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools Operate Mark Regexp Immediate Subdir Help

/home/rjd4/EmacsCourse:
used 362 available 16777212
drwxr-xr-x  1 rjd4 rjd4    512 2006-03-06 16:03 .
drwxr-xr-x  1 rjd4 rjd4    512 2006-03-06 14:37 ..
-rw-r--r--  1 rjd4 rjd4  2230 2006-03-06 14:37 iterator1.c
-rw-r--r--  1 rjd4 rjd4    693 2006-03-06 14:37 iterator2.f
-rw-r--r--  1 rjd4 rjd4    418 2006-03-06 14:37 Makefile
-rw-r--r--  1 rjd4 rjd4    102 2006-03-06 14:37 temp.gplt
-rw-r--r--  1 rjd4 rjd4 364748 2006-03-06 14:37 treasure.txt

-u:%% EmacsCourse (Dired by name)--L5--All-----
```

Summary of commands

Key sequence

C-x b

C-x C-b

C-x o ("oh")

C-x 1 ("one")

Operation

Switch to most recent other **b**uffer

Split window and show list of **b**uffers

Switch to **o**ther half window

Return to displaying **1** buffer

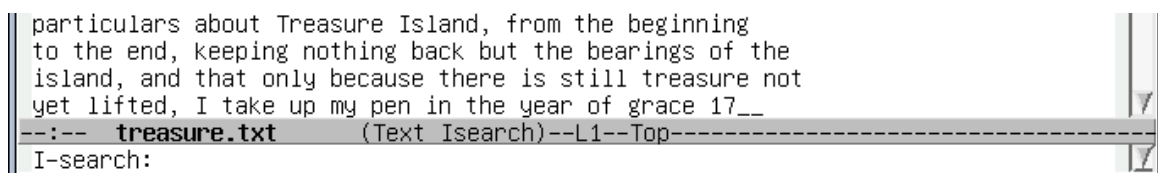
Searching and replacement

We will move into the `treasure.txt` buffer for this part of the course. The principles are the same for all forms of text file and we will benefit from having a large file to work in. All of the previously described movement operations still apply but we can also move by word, sentence, or paragraph.

Key sequence	Short cut	Operation
C-b	Left	Move cursor left (b ackwards) one character
C-f	Right	Move cursor right (f orwards) one character
C-n	Down	Move cursor down one row (to n ext row)
C-p	Up	Move cursor up one row (to p revious row)
C-a	Home	Move cursor to start of line
C-e	End	Move cursor to e nd of line
C-x C-s		Save the file
M-<		Move to start of buffer
M->		Move to end of buffer
C-v	PageDown	Move one screenful down the buffer
M-v	PageUp	Move one screenful up the buffer
M-b		Move cursor b ackwards one word
M-f		Move cursor f orwards one word
M-a		Move cursor backwards one sentence
M-e		Move cursor forwards one sentence
M-{		Move cursor forwards one paragraph
M-}		Move cursor backwards one paragraph

(Recall that some of the control characters are shifted characters in their own rights: "<", ">", "{", "}".)

However, even these are not much use in a file of 8,738 lines. We need to be able to move about in the file by searching for text. We begin with the cursor at the start of the first paragraph and search for the string "Benbow". To start the search we type C-s. The mini-buffer changes to reflect that we are searching and will show us what we are searching for:



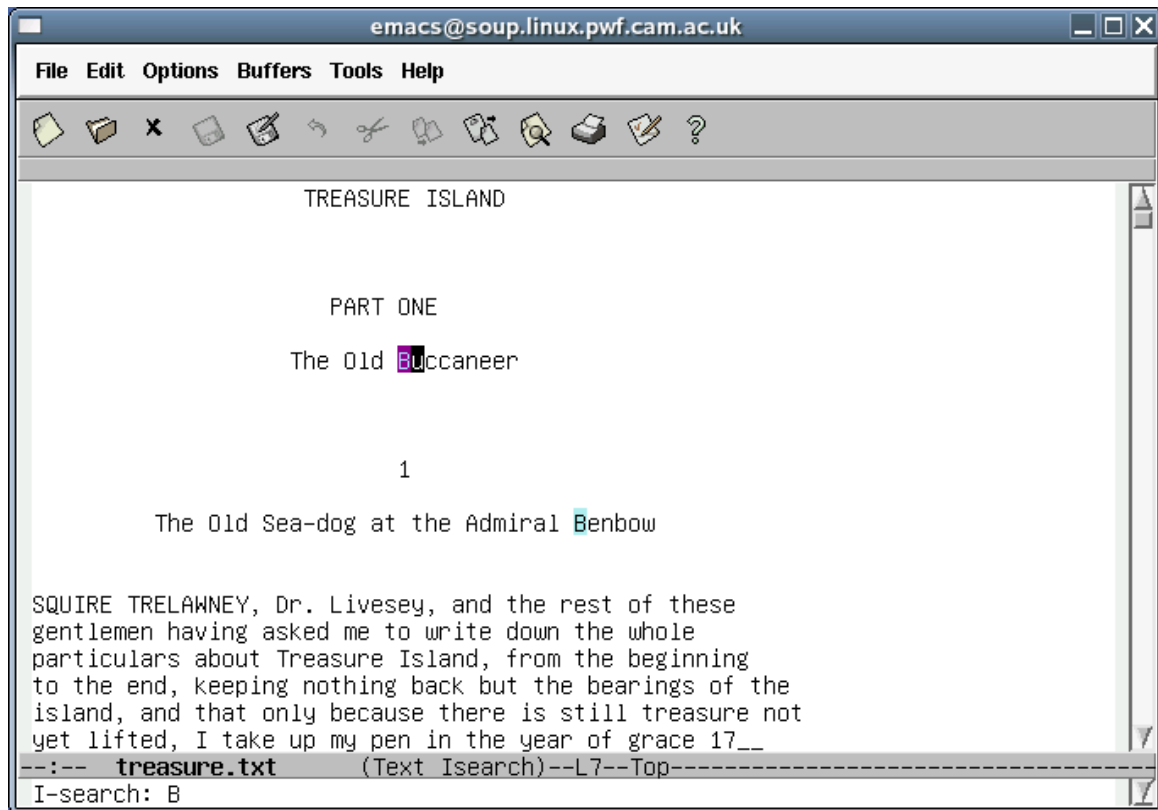
```

particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text Isearch)--L1--Top-----
I-search:

```

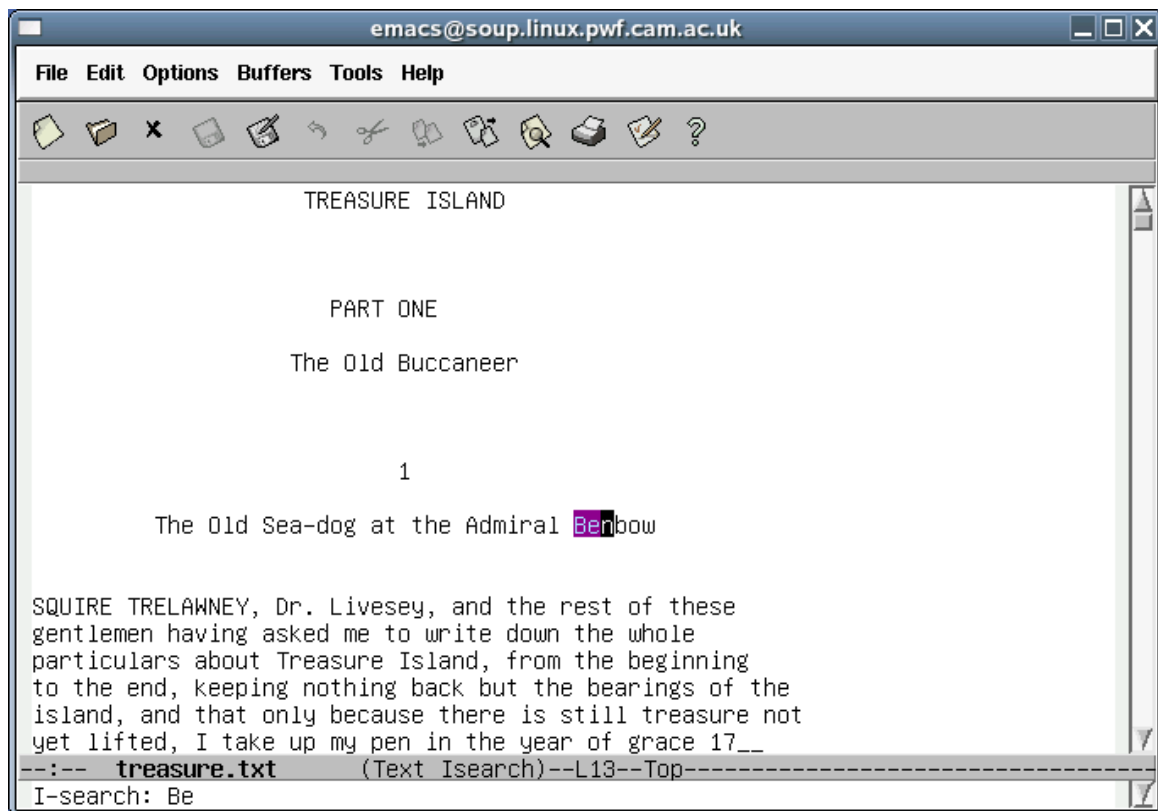
Introduction to Emacs

The prompt “I-search” means “interactive search”. Once we type the “B” of “Benbow” we see various things happen:

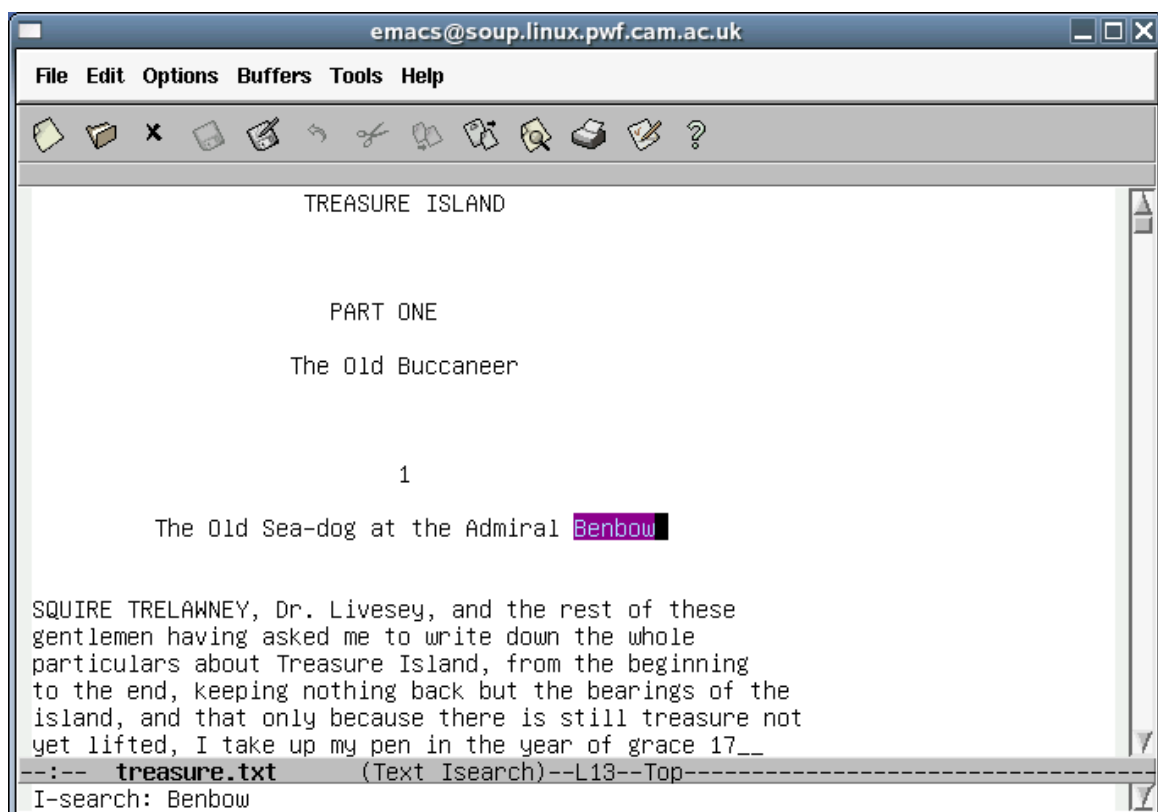


The “B” in “Buccaneer” has been highlighted in purple. The following “B” in “Benbow” has been highlighted in blue. The cursor has been moved to follow the first match (in “Buccaneer”). The reaction has happened without any pressing of Return.

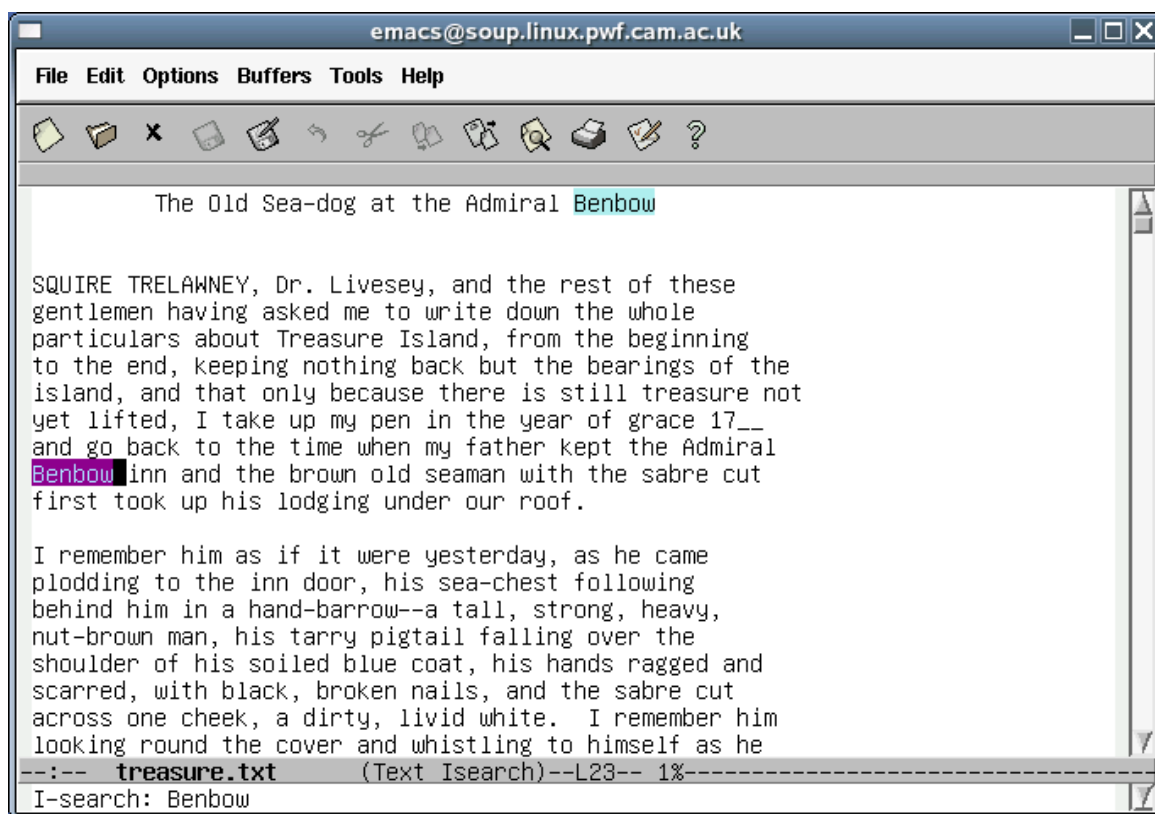
Now we press the “e” as the second letter of “Benbow”.



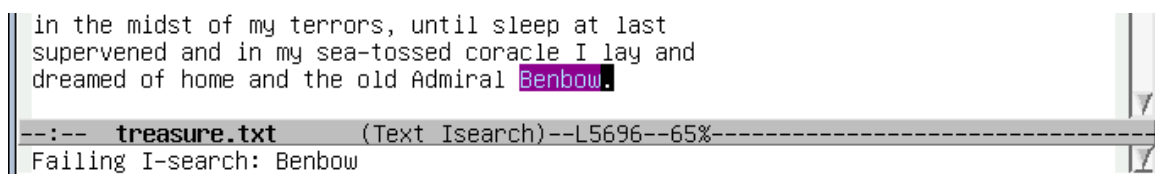
The cursor, and the purple highlighting has jumped to the instance in the chapter title. If i type the rest of the word the purple highlighting extends to the entire word.



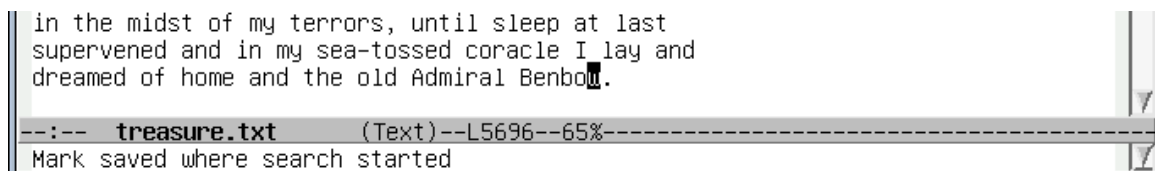
If I press C-s again, then the cursor, and the purple highlighting jumps to the next match:



Note how the previous match is now marked in blue. This is the general case: the current match, which has the cursor, is marked in purple. All other matches are marked in blue. If we run through the various instances of “Benbow” in the text¹⁸ we eventually reach the last one. If we press C-s again, Emacs beeps and the mini-buffer reports that the search has failed to find another match:



It's all very well finding text. Once we have found our way to an appropriate part of a buffer we want to break out of the search and to do something else. The simplest way is just to move the cursor. Any cursor arrow will move the cursor and break out of the interactive search. If we press Left here then we leave search and the cursor moves to the “w” of “Benbow”:



Note the report in the mini-buffer: “Mark saved where search started”. We will return to this when we talk about marking chunks of text for processing.

As well as searching forwards with C-s we can search backwards with C-r. Starting at our

¹⁸There are 20.

position on the final “Benbow” of the document we can search backwards for previous instances of “Admiral” in an exactly analogous way to how we searched forwards for “Benbow”:

```

in the midst of my terrors, until sleep at last
supervened and in my sea-tossed coracle I lay and
dreamed of home and the old Admiral Benbow.
--:-- treasure.txt (Text Isearch)--L5696--65%-----
I-search backward: Admiral

```

Note that the cursor appears at the start of the matching text and not at the end. We can switch direction in mid-search. If we search backwards a few times with repeated C-r we can turn round and start searching forwards by pressing C-s instead of C-r. The first time we switch direction all that happens is that the cursor jumps to the other end of the matched text. Subsequent searches move the cursor to the next matching word.

We can also replace the text we have located with alternative text. If we want to butcher “Treasure Island” by replacing all instances of “Admiral Benbow” with “Admiral Nelson” then we can.

We start by moving to the beginning of the buffer with M-<¹⁹. Then we press M-%²⁰:

```

particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Query replace:

```

The mini-buffer shows the “Query replace:” prompt. We enter the text we want replaced, “Benbow”, and press Return:

```

particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Query replace Benbow with:

```

We are now prompted for the text we want to replace it with, “Nelson”:

```

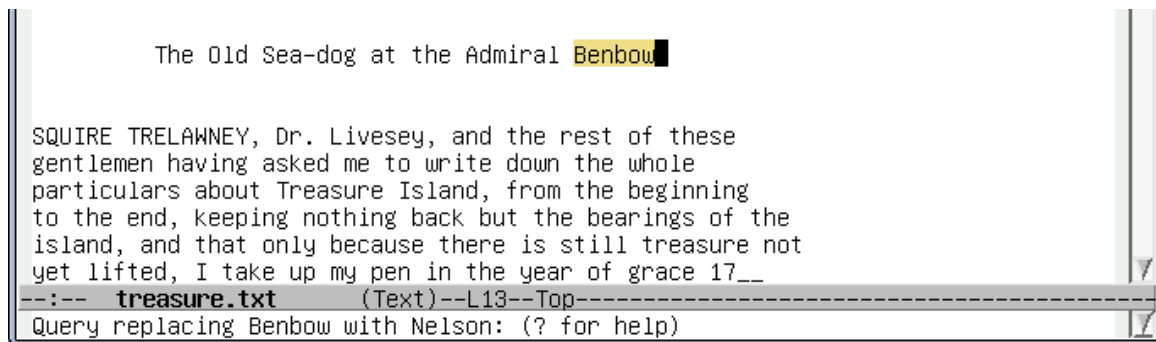
particulars about Treasure Island, from the beginning
to the end, keeping nothing back but the bearings of the
island, and that only because there is still treasure not
yet lifted, I take up my pen in the year of grace 17__
--:-- treasure.txt (Text)--L1--Top-----
Query replace Benbow with: Nelson

```

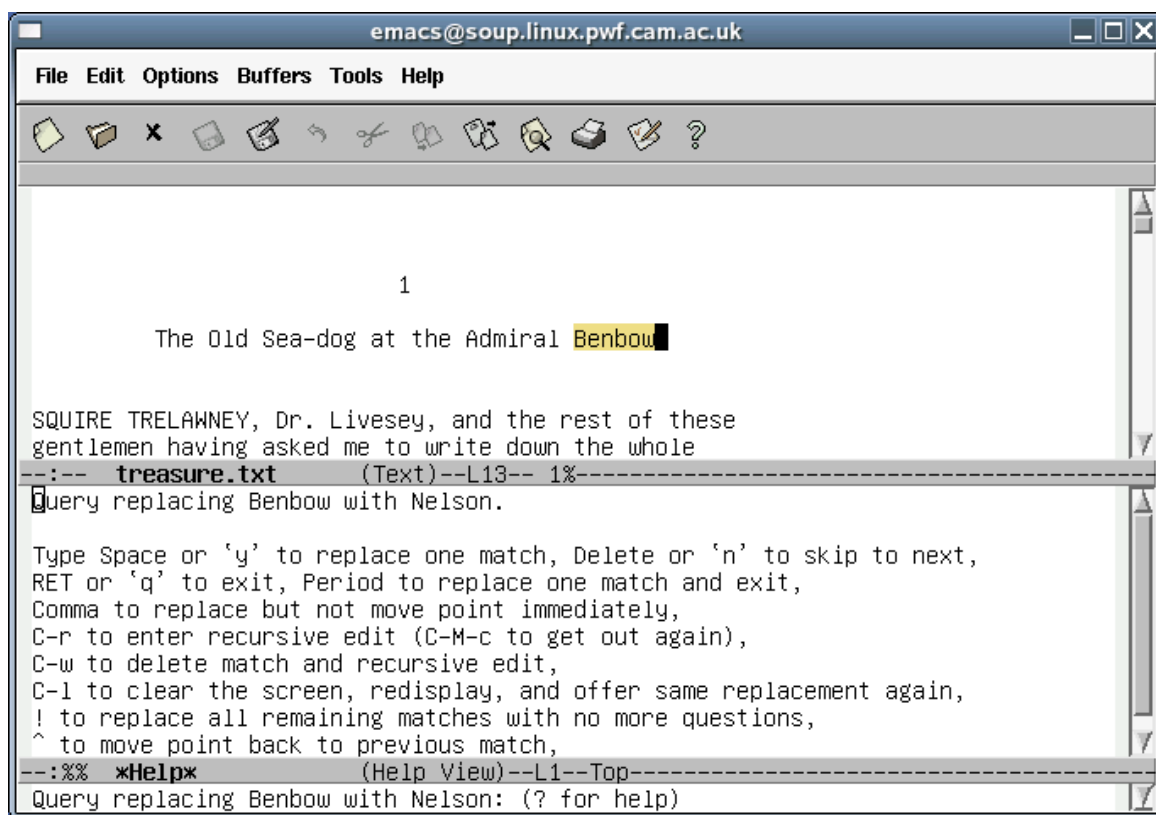
¹⁹The “less than” character is a Shift-comma on a UK PC keyboard.

²⁰The percent character is Shift-5 on a UK PC keyboard.

and press Return again:



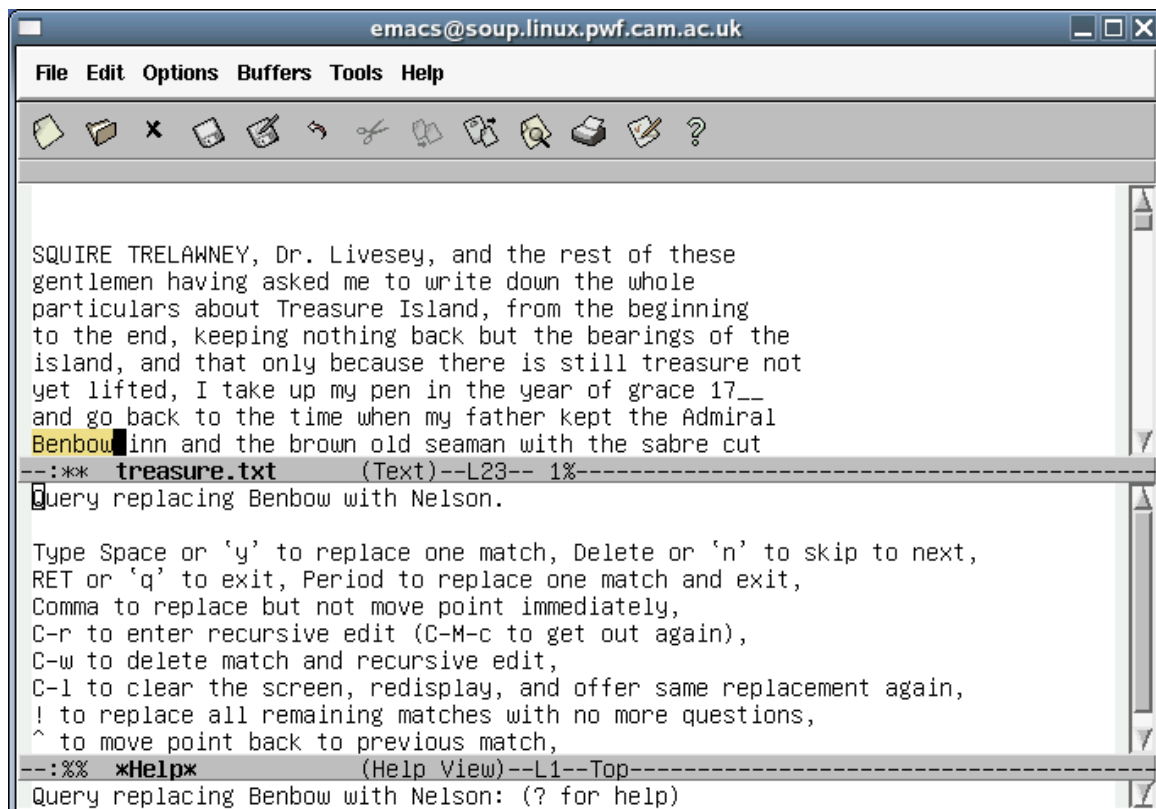
The cursor jumps forwards to the first matching text and highlights it. It will be instructive to ask for help here as offered at the prompt. We will press “?”:



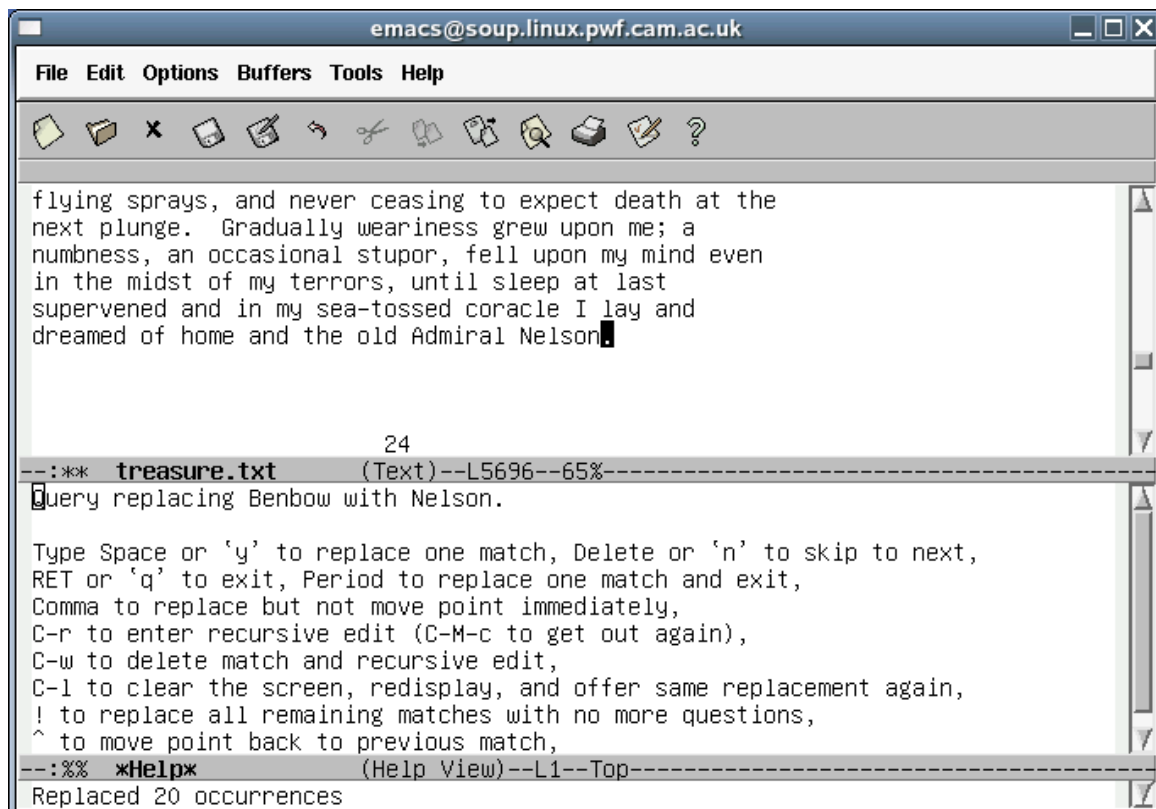
The option to change the text is to press the space bar (hereafter referred to as Space). The most commonly useful options are

Key	Alternative	Action
y	Space	Change this instance and move on to the next.
n		Don't change this instance but move to the next.
q	Return	Don't change this and quit.
!		Change this instance and all future instances without prompting.

We will press Space to change this instance and move on. Note that we are still stuck with the split screen presentation of the help text.



This time we will just press "!" to complete the global exchange.



The cursor is left at the end of the last change and the total number of changes is reported in the mini-buffer. The cursor still resides in the window corresponding to the text, so pressing C-x 1 returns us to single-window mode. Generally, whenever Emacs returns to single window mode from multi-window mode the window that “wins” (i.e. the one that gets shown) is the one that had the live cursor in it at the time.

If you find yourself performing substitutions with the same values there is a useful trick which can save you typing. When being prompted for values you can press Up (or Down) to move through the history of the various strings used in the substitutions, as either the substituted or substituting text.

Summary of commands

Key sequence

C-s

C-r

M-%

suboptions: **y**
n

q
!

Space

Return

Operation

Interactive search forwards

Interactive search backwards

Search (forwards) and replace

Change this instance and move on to the next.
Don't change this instance but move to the next.

Don't change this and quit.

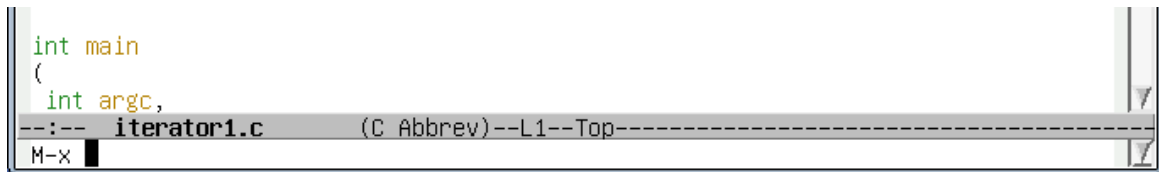
Change this instance and all future instances without prompting.

Jumping to a line

For this section we will move back to our C program, `iterator1.c`.²¹ Move to the top of the buffer²² just so that we are all seeing the same thing.

Suppose we want to move the cursor to line 48, perhaps because a compiler has identified an error there. Emacs does have a command to move to a particular line. It's called `goto-line` and does exactly what it says on the tin. What Emacs does not have, though, is a simple key sequence to trigger this command. So we need to know how to issue arbitrary Emacs commands.

To issue an arbitrary Emacs command, type `M-x`. The mini-buffer will be ready to receive the command:

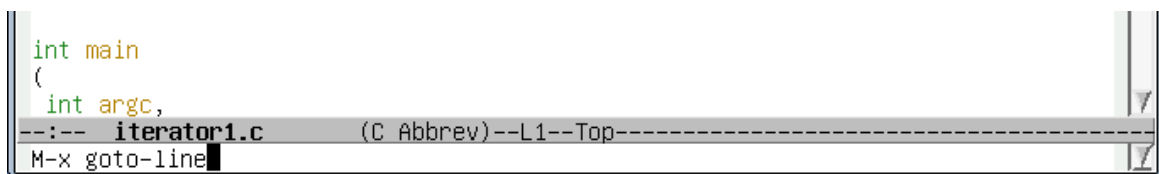


```

int main
(
  int argc,
--:-- iterator1.c      (C Abbrev)--L1--Top-----
M-x

```

We can now type the name of the command, `goto-line`:

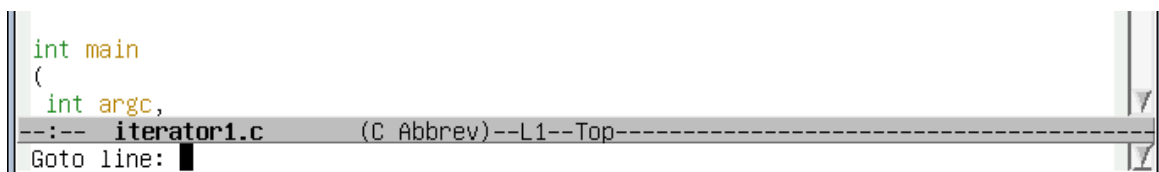


```

int main
(
  int argc,
--:-- iterator1.c      (C Abbrev)--L1--Top-----
M-x goto-line

```

and press Return to issue it. It now prompts for the line number to go to:

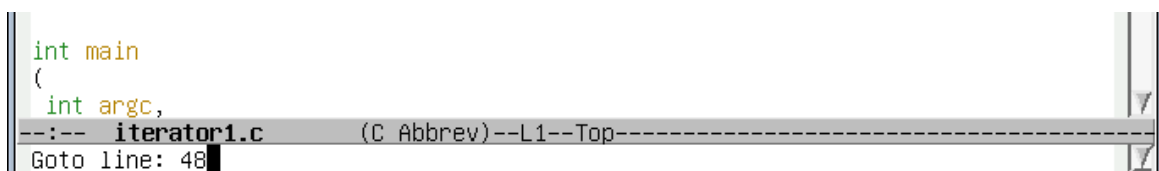


```

int main
(
  int argc,
--:-- iterator1.c      (C Abbrev)--L1--Top-----
Goto line:

```

which we give



```

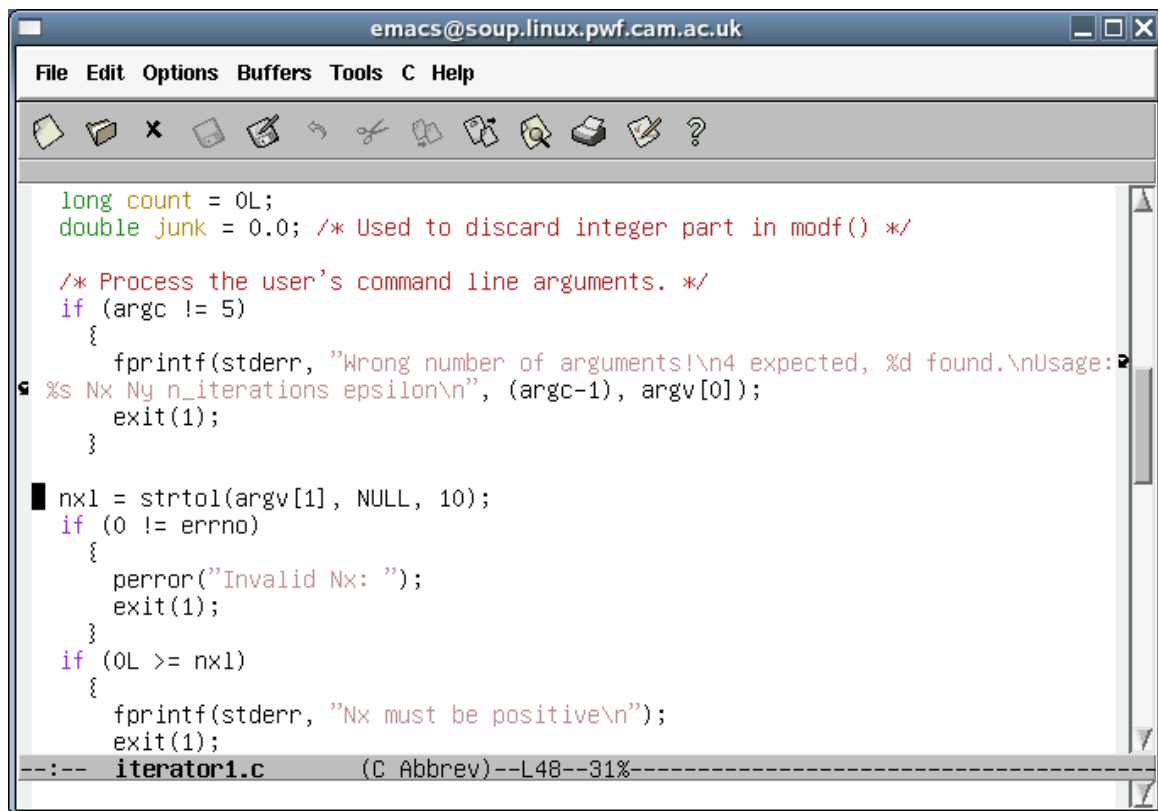
int main
(
  int argc,
--:-- iterator1.c      (C Abbrev)--L1--Top-----
Goto line: 48

```

²¹ `C-x b` to switch buffer

²² `M-<`

and press Return:



```

emacs@soup.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools C Help

long count = 0L;
double junk = 0.0; /* Used to discard integer part in modf() */

/* Process the user's command line arguments. */
if (argc != 5)
{
    fprintf(stderr, "Wrong number of arguments!\n4 expected, %d found.\nUsage: %s Nx Ny n_iterations epsilon\n", (argc-1), argv[0]);
    exit(1);
}

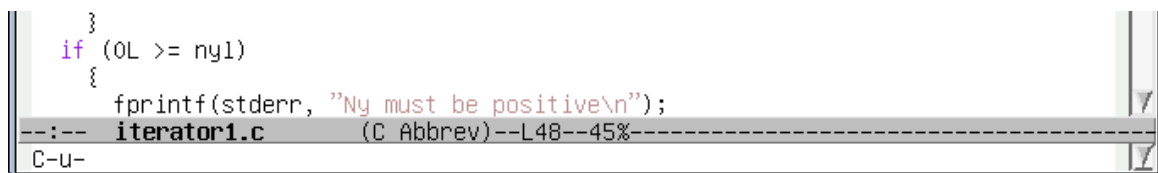
nx1 = strtol(argv[1], NULL, 10);
if (0 != errno)
{
    perror("Invalid Nx: ");
    exit(1);
}
if (0L >= nx1)
{
    fprintf(stderr, "Nx must be positive\n");
    exit(1);
}
--:-- iterator1.c (C Abbrev)--L48--31%-----

```

The cursor jumps to the start of the selected line.

There is an alternative way of getting the 48 through to the goto-line command. We will use this alternative mechanism to jump to line 60.

We start by entering C-u:



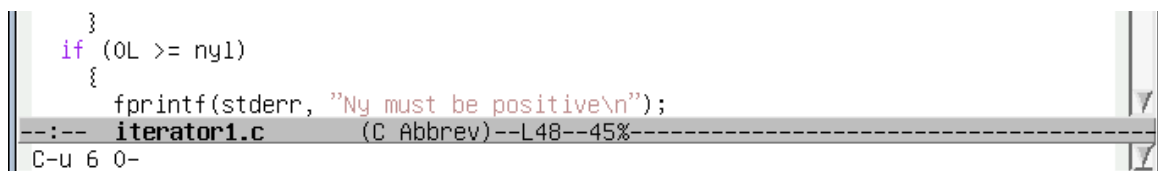
```

}
if (0L >= ny1)
{
    fprintf(stderr, "Ny must be positive\n");
}
--:-- iterator1.c (C Abbrev)--L48--45%-----
C-u-

```

(Actually the "C-u-" prompt only appears if we don't type anything for a couple of seconds.)

Then we enter 60. We do *not* press Return.



```

}
if (0L >= ny1)
{
    fprintf(stderr, "Ny must be positive\n");
}
--:-- iterator1.c (C Abbrev)--L48--45%-----
C-u 60-

```

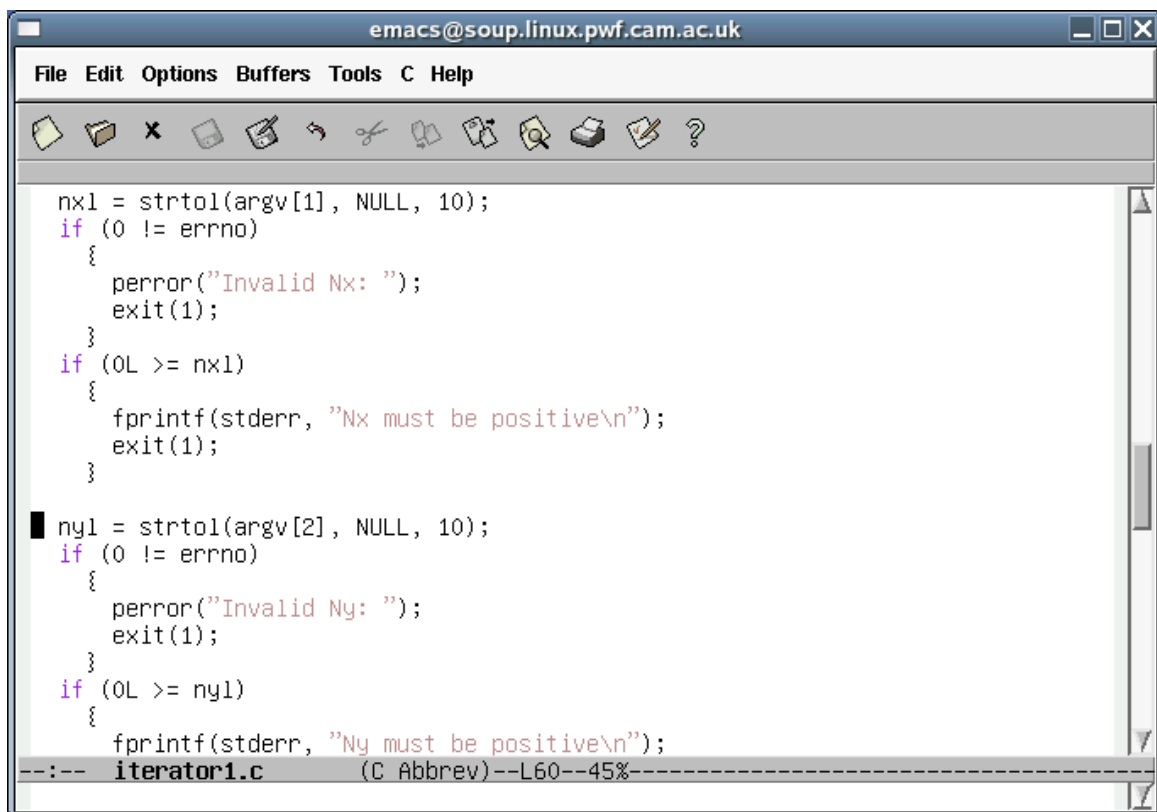
Next, we enter M-x for the command:

```
    }  
    if (0L >= ny1)  
    {  
        fprintf(stderr, "Ny must be positive\n");  
--:-- iterator1.c (C Abbrev)--L48--45%-----  
60 M-x
```

Note how the 60 appears as a prefix for the command. We can now enter our command:

```
    }  
    if (0L >= ny1)  
    {  
        fprintf(stderr, "Ny must be positive\n");  
--:-- iterator1.c (C Abbrev)--L48--45%-----  
60 M-x goto-line
```

and press Return for it to be executed:

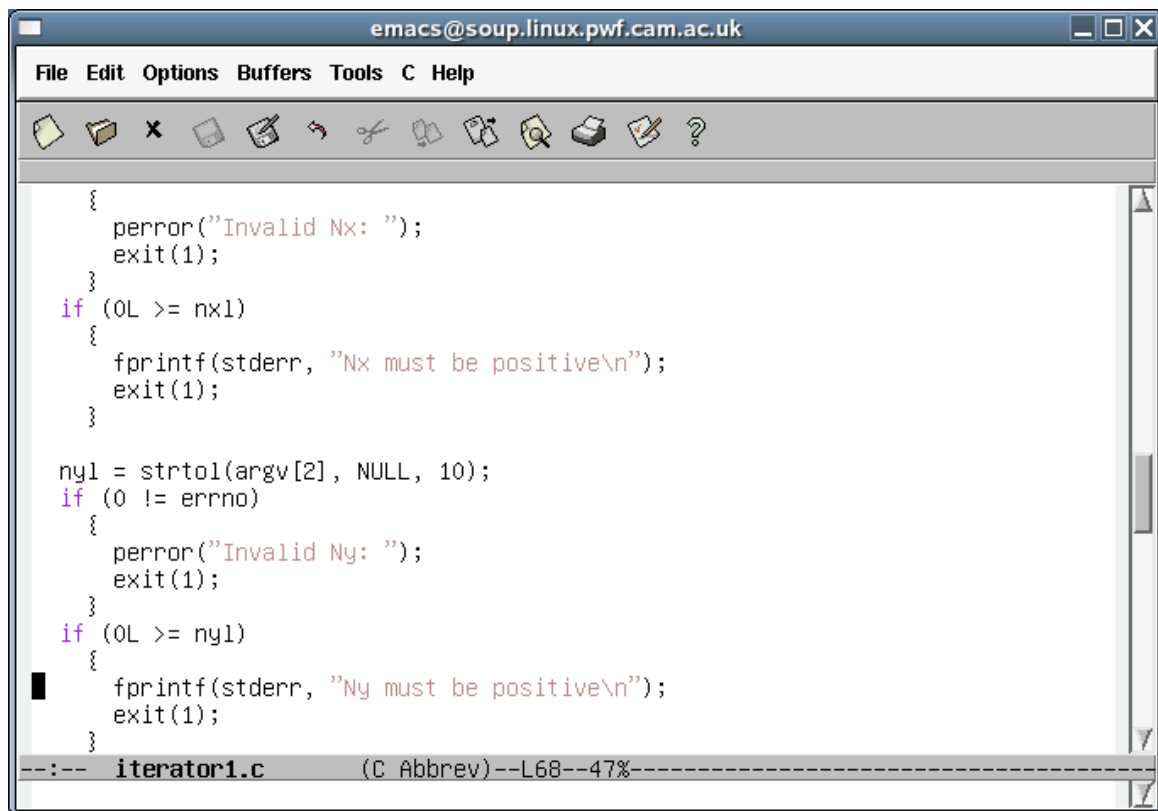


The screenshot shows the Emacs editor window titled 'emacs@soup.linux.pwf.cam.ac.uk'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'C', and 'Help'. The toolbar contains various icons for file operations and editing. The main text area displays C code from 'iterator1.c'. The cursor is at line 60, and the command 'goto-line' has been entered. The status bar at the bottom shows '--:-- iterator1.c (C Abbrev)--L60--45%-----'.

```
nx1 = strtol(argv[1], NULL, 10);  
if (0 != errno)  
{  
    perror("Invalid Nx: ");  
    exit(1);  
}  
if (0L >= nx1)  
{  
    fprintf(stderr, "Nx must be positive\n");  
    exit(1);  
}  
ny1 = strtol(argv[2], NULL, 10);  
if (0 != errno)  
{  
    perror("Invalid Ny: ");  
    exit(1);  
}  
if (0L >= ny1)  
{  
    fprintf(stderr, "Ny must be positive\n");  
--:-- iterator1.c (C Abbrev)--L60--45%-----
```

This may seem unnecessarily complex but it illustrates this way of getting numerical arguments into commands. It comes into its own for commands where the argument is optional.

Suppose we move the cursor to a line near the top of bottom of the window:

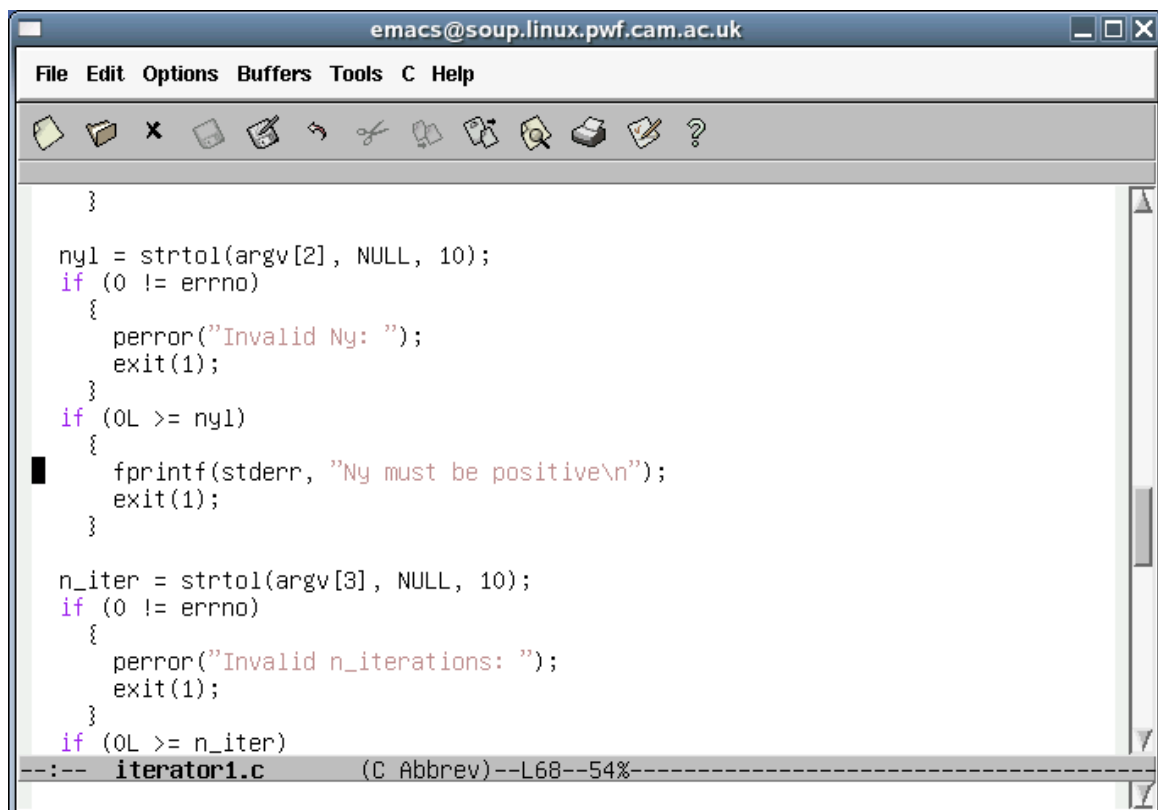


The screenshot shows the Emacs editor window titled "emacs@soup.linux.pwf.cam.ac.uk". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". The toolbar contains various icons for file operations and editing. The code in the buffer is a C program with several error handling blocks. The cursor is positioned at the end of the line "exit(1);". The status bar at the bottom indicates the file is "iterator1.c", the cursor is at column 47, line 68, and the buffer is "(C Abbrev)".

```
{
    perror("Invalid Nx: ");
    exit(1);
}
if (OL >= nx1)
{
    fprintf(stderr, "Nx must be positive\n");
    exit(1);
}

ny1 = strtol(argv[2], NULL, 10);
if (0 != errno)
{
    perror("Invalid Ny: ");
    exit(1);
}
if (OL >= ny1)
{
    fprintf(stderr, "Ny must be positive\n");
    exit(1);
}
}
--:-- iterator1.c (C Abbrev)--L68--47%
```

and then type C-l ("ell"):



The screenshot shows the same Emacs editor window. The cursor has moved to the end of the line "fprintf(stderr, "Ny must be positive\n");". The status bar at the bottom indicates the file is "iterator1.c", the cursor is at column 54, line 68, and the buffer is "(C Abbrev)".

```
}

ny1 = strtol(argv[2], NULL, 10);
if (0 != errno)
{
    perror("Invalid Ny: ");
    exit(1);
}
if (OL >= ny1)
{
    fprintf(stderr, "Ny must be positive\n");
    exit(1);
}

n_iter = strtol(argv[3], NULL, 10);
if (0 != errno)
{
    perror("Invalid n_iterations: ");
    exit(1);
}
if (OL >= n_iter)
{
    fprintf(stderr, "n_iter must be positive\n");
    exit(1);
}
}
--:-- iterator1.c (C Abbrev)--L68--54%
```

The line with the cursor on it jumps to the middle of the window. This is the default behaviour

for C-l: it jumps the current line to the middle of the screen. However, if it is given a number it will jump the cursor's line to that line within the screen. The argument 0 means "first line" so the sequence "C-u 0 C-l" (as it would be written in Emacs notation) will jump the cursor's line to the top of the screen:

```

    fprintf(stderr, "Ny must be positive\n");
    exit(1);
}

n_iter = strtol(argv[3], NULL, 10);
if (0 != errno)
{
    perror("Invalid n_iterations: ");
    exit(1);
}
if (0L >= n_iter)
{
    fprintf(stderr, "n_iterations must be positive\n");
    exit(1);
}

epsilon = strtod(argv[4], NULL);
if (0 != errno)
{
    perror("Invalid epsilon: ");
    exit(1);
}

--:-- iterator1.c (C Abbrev)--L68--60%-----
C-u 0 C-l

```

A word of caution is required about C-u. If it precedes an "insertion command" (and as far as Emacs is concerned typing a letter to insert that letter in the text is such a command) then it triggers as many repeats as the number passed to C-u. So "C-u 48 g" will insert 48 letters "g".

Summary of commands

Key sequence / Command

M-x goto-line

Operation

Prompt for a line to move to and then move to it.

C-u N M-x goto-line

Move to line *N*.

C-l

Centre current line in window.

C-u N C-l

Move current line to line *N* in window
(0 = top line)

C-u N x

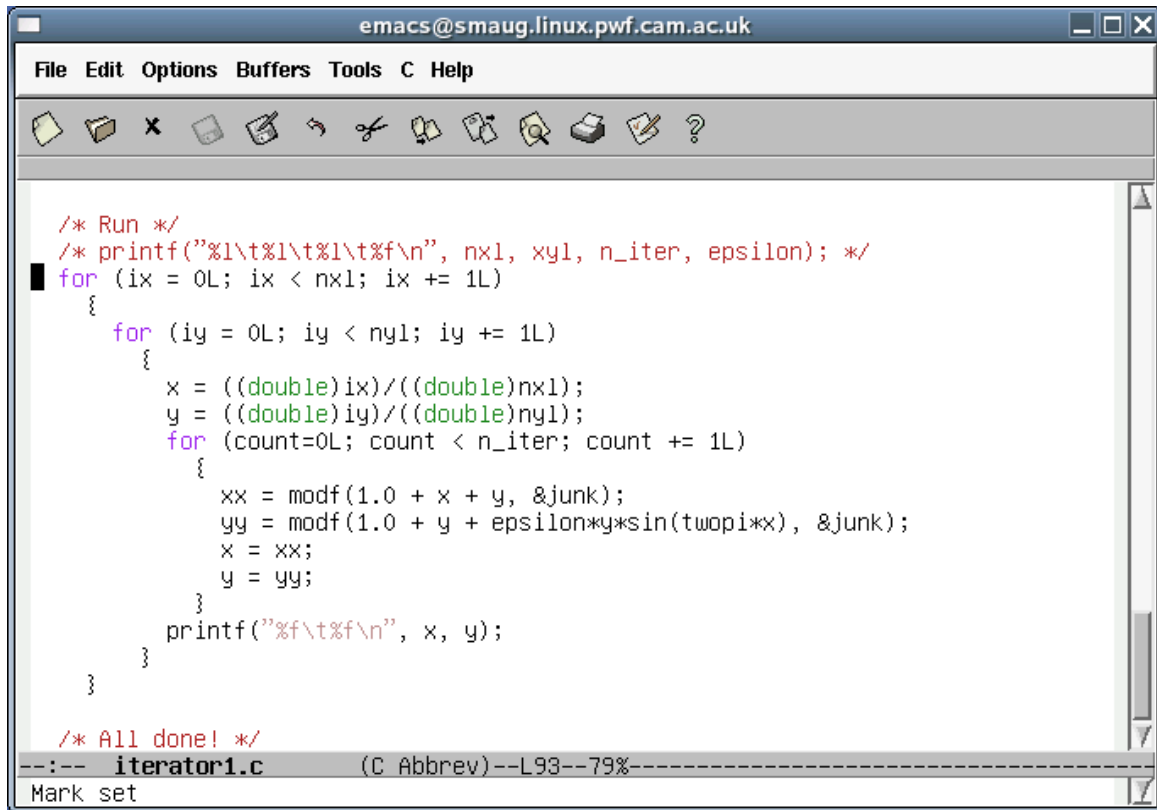
Insert *N* many letters *x*.

Copying, Cutting, and Pasting

A standard operation in all text editors is to be able to copy or cut (copy and remove) a block of text and then to paste (insert) it zero or more times elsewhere in the document.

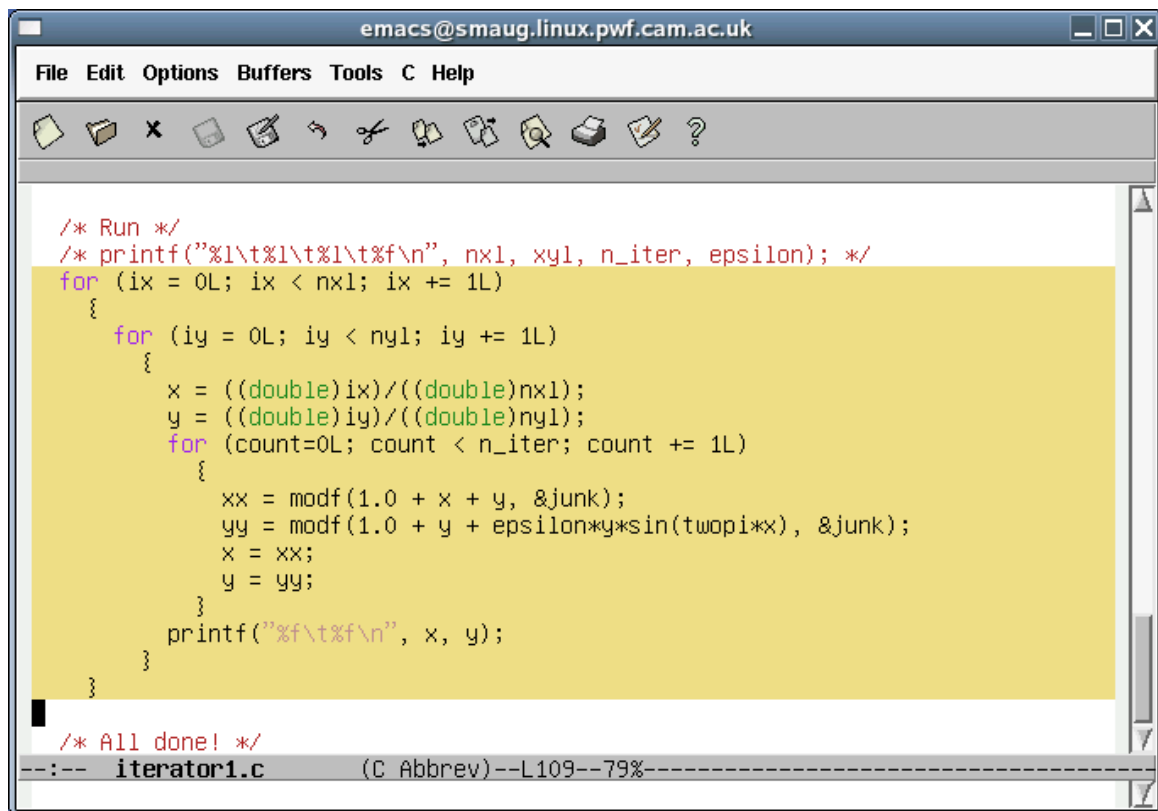
The copy/cut and paste model in Emacs is straightforward. We mark an area of text to be operated on, we copy or cut it and then we paste it somewhere else.

To mark a block of text we have a number of means we can use. We start by identifying the start of the area. To do this we move the cursor to the appropriate location and press C-SPC²³.



The mini-buffer confirms the action by reporting that the “mark” has been set. The mark is the point at the start of the area being defined. Next we move the cursor to the end of the text block we are interested in. The defined area is shaded. (A mark is set by other operations also but the shading is specific to this sort of operation.)

²³ SPC is how the Emacs keystroke notation represents the Space bar on the keyboard.



```

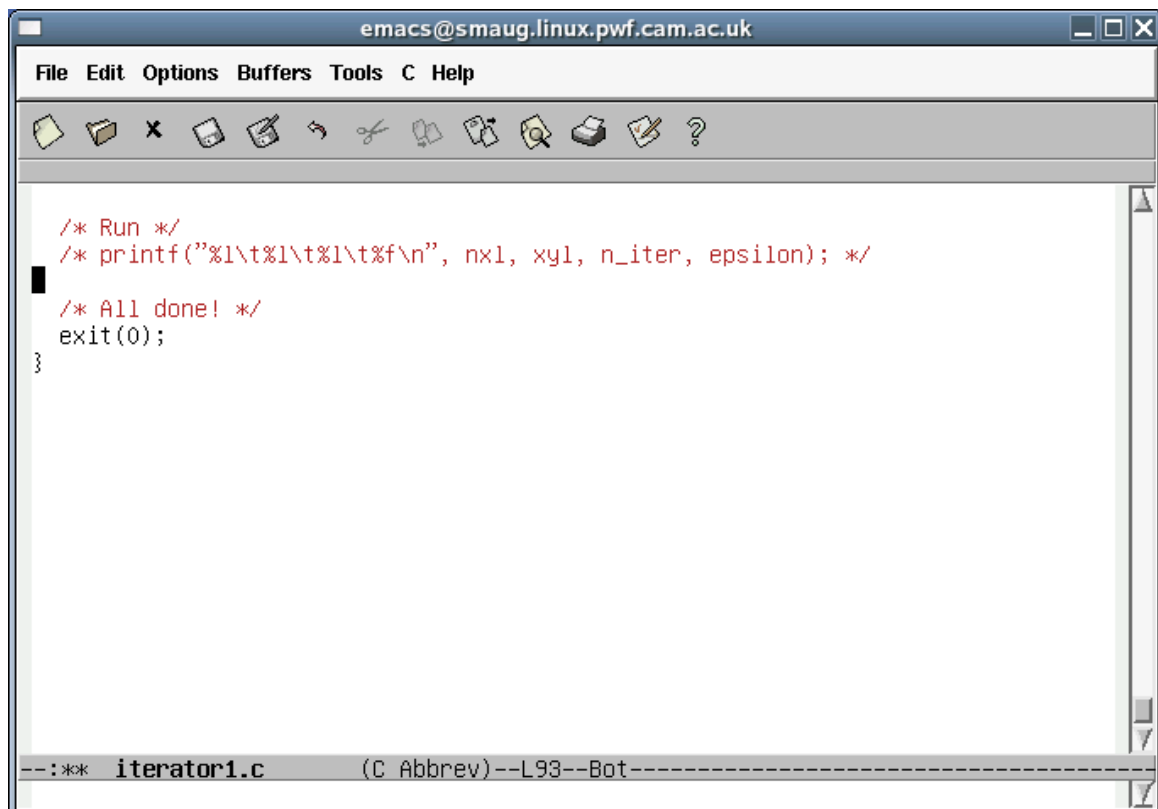
emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools C Help

/* Run */
/* printf("%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon); */
for (ix = 0L; ix < nx1; ix += 1L)
{
    for (iy = 0L; iy < ny1; iy += 1L)
    {
        x = ((double)ix)/((double)nx1);
        y = ((double)iy)/((double)ny1);
        for (count=0L; count < n_iter; count += 1L)
        {
            xx = modf(1.0 + x + y, &junk);
            yy = modf(1.0 + y + epsilon*y*sin(twopi*x), &junk);
            x = xx;
            y = yy;
        }
        printf("%f\t%f\n", x, y);
    }
}

/* All done! */
--:-- iterator1.c (C Abbrev)--L109--79%

```

To cut the marked block, we would press C-w. To copy it we press M-w. In this example we will cut the text.



```

emacs@smaug.linux.pwf.cam.ac.uk
File Edit Options Buffers Tools C Help

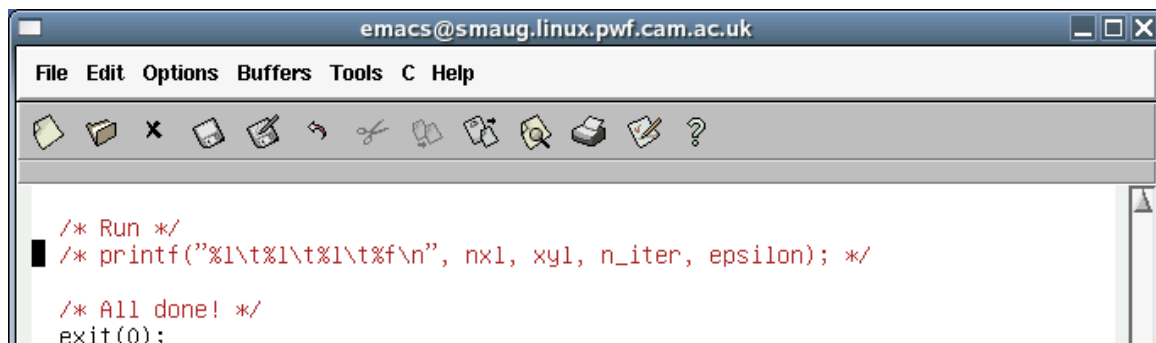
/* Run */
/* printf("%1\t%1\t%1\t%f\n", nx1, xy1, n_iter, epsilon); */

/* All done! */
exit(0);
}

--:** iterator1.c (C Abbrev)--L93--Bot

```

We can now paste the text in wherever the cursor is by pressing C-y. In this case we will simply move the cursor up one line:

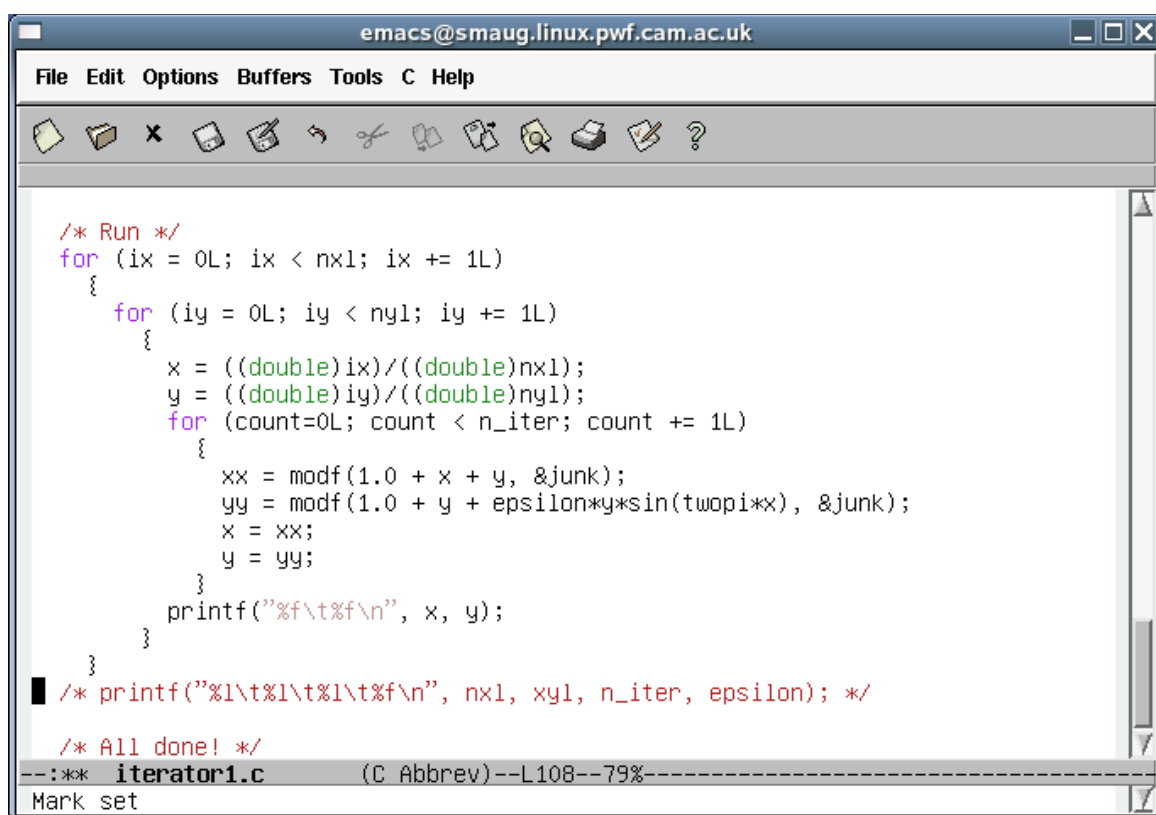


The screenshot shows the Emacs editor window titled 'emacs@smaug.linux.pwf.cam.ac.uk'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'C', and 'Help'. The toolbar contains various icons for file operations. The text area shows a C program snippet with the following code:

```
/* Run */
/* printf("%1\t%1\t%1\t%f\n", nx1, xyl, n_iter, epsilon); */

/* All done! */
exit(0);
```

and then press C-y to paste back the material:



The screenshot shows the Emacs editor window with the same title and menu bar. The text area now contains a more complete C program. The code is as follows:

```
/* Run */
for (ix = 0L; ix < nx1; ix += 1L)
{
    for (iy = 0L; iy < nyl; iy += 1L)
    {
        x = ((double)ix)/((double)nx1);
        y = ((double)iy)/((double)nyl);
        for (count=0L; count < n_iter; count += 1L)
        {
            xx = modf(1.0 + x + y, &junk);
            yy = modf(1.0 + y + epsilon*y*sin(twopi*x), &junk);
            x = xx;
            y = yy;
        }
        printf("%f\t%f\n", x, y);
    }
}
/* printf("%1\t%1\t%1\t%f\n", nx1, xyl, n_iter, epsilon); */

/* All done! */
```

At the bottom of the window, the status bar shows the file name 'iterator1.c', the mode '(C Abbrev)', the line number '--L108--', and the percentage '79%'. Below the status bar, it says 'Mark set'.

Summary of commands

Key sequence

C-SPC

C-w

M-w

C-y

Operation

Mark the start of the region to be copied/cut.

Cut the marked region to the cut buffer.

Copy the marked region to the cut buffer.

Paste ("yank") the cut buffer into the document.

Keystroke Macros

We will load the file `action.c` and look specifically at its `switch()` statement.

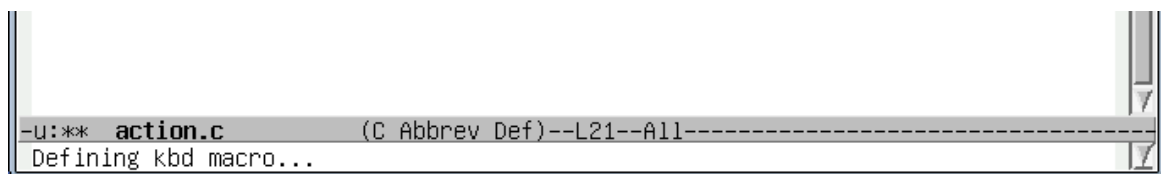
```
switch (op_code)
{
  case(0):
  {
    printf("Zero: %s\n", string);
    break;
  }
  case(1):
  {
    printf("One: %s\n", string);
    break;
  }
  default:
  {
    fprintf(stderr, "Unknown op code: %d\n", op_code);
    exit(1);
  }
}
```

Adding another block for `op_code "2"` involves a lot of replication. It is possible to reduce this with judicious use of cut and paste, but here's another approach that introduces a very powerful mechanism.

We start by adding a "2", which will be the new `op_code`, with the cursor over it:

```
case(1):
{
  printf("One: %s\n", string);
  break;
}
2
default:
```

Next, we issue the command `C-x (`. The mini-buffer reports that we are now defining a "keyboard macro".



Next, we very carefully build the `case()` block for the number 2.

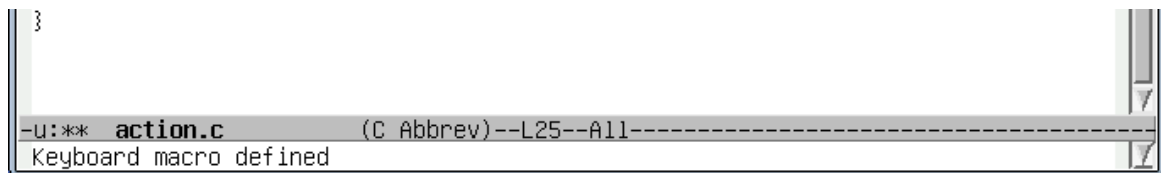
You will have noticed by now that each time you try to make Emacs do something it can't it beeps²⁴ at you. If you try to search for a string that's not there it beeps; if you try to go beyond the start or end of the file it beeps; if you issue a command it doesn't recognise it beeps. We need to be careful not to make it beep while defining the macro.

²⁴ This beeping is called "ringing the bell", a name that dates back to the days when teletype terminals had real bells on them. Some systems are configured to flash the screen at you rather than make any sound at all. This is called a "visual bell".

To pull off a trick later on, we will leave the cursor at the start of the line immediately below the block we have just defined. This is the start of the “default:” line in our case.

```
case(1):
{
    printf("One: %s\n", string);
    break;
}
case(2):
{
    break;
}
default:
```

and then press C-x). The mini-buffer confirms that it has recorded a keyboard macro:



The screenshot shows the Emacs mini-buffer at the bottom of the window. The buffer contains the text: `-u:** action.c (C Abbrev)--L25--A11-----` followed by a dashed line and the text `Keyboard macro defined`. The buffer is highlighted with a grey background.

Now, to see what we have achieved, put a 3 on a line to itself after the 2-block with the cursor over it, exactly as it was at the start of defining the 2-block.

```
case(1):
{
    printf("One: %s\n", string);
    break;
}
case(2):
{
    break;
}
3
default:
```

Now we press C-X e to execute the keyboard macro we have defined.

Exactly the same keys we pressed to turn the 2 into the 2-block have now been applied to the 3:

```
case(1):
{
    printf("One: %s\n", string);
    break;
}
case(2):
{
    break;
}
case(3):
{
    break;
}
default:
```

Now we can also see what was useful about having the cursor left at the start of the next line at the end of our keyboard macro. We can add a column of numbers

```
case(2):
{
    break;
}
case(3):
{
    break;
}
4
5
6
7
8
9
default:
```

and each time we press C-x e the next number is converted and the cursor is left in place ready to convert the next.

```
case(2):
{
    break;
}
case(3):
{
    break;
}
case(4):
{
    break;
}
case(5):
{
    break;
}
6
7
8
9
default:
```

If there are four numbers left we can also issue the command C-u 4 C-x e to run it four times.

```
case(6):
{
    break;
}
case(7):
{
    break;
}
case(8):
{
    break;
}
case(9):
{
    break;
}
default:
```

And now a word of caution. Suppose we try the same trick with 10. Depending on exactly what you typed in the keystroke macro it may fail. Suppose I start like this:

```
case(9):  
  {  
    break;  
  }  
10  
default:
```

then I may end up with this:

```
case(9):  
  {  
    break;  
  }  
case(1):  
  {  
    break;  
  }0  
default:
```

So what went wrong?

The problem is that when I was defining my keyboard macro I moved the cursor beyond the 2 is was setting up a block for by pressing the Right key once. This was fine for all other single-digit numbers, but failed for 10.

When defining a keyboard macro it is important to remember why you are moving the cursor. If you are moving it to the end of a line (as I was) then it is important that you use C-e or End and not simply press the Right key enough times for that particular case. If the number is different for other cases then things will fail.

Summary of commands

Key sequence

C-x (

C-x)

C-x e

C-u N C-x e

Operation

Start defining a keyboard macro.

End of definition of keyboard macro.

Execute keyboard macro.

Execute keyboard macro *N* times.

Recap and conclusion

Hopefully by the end of this course you will have learnt the following points. However, there is no substitute for experience. Try using Emacs for a month and get your fingers used to it. You may never go back!

- Launching Emacs.
- Quitting Emacs.
- Loading files into buffers.
- Saving buffers back into files.
- Navigation within a buffer by character, word, line or to the ends of the buffer.
- Searching forwards and backwards in Emacs.
- Editing plain text files.
- Getting assistance from Emacs editing a program.
- Managing multiple buffers and switching between them.
- Viewing two buffers at the same time and switching between them.
- Moving directly to a particular line by line number.
- Copying and pasting within and between buffers.
- Creating and using keyboard macros.

Summary of key strokes

This table summarizes the key sequences in this course, in the rough order they first appear.

Key sequence	Short cut	Operation
C-x C-c		Exit Emacs
C-_		Undo last edit
C-g		Quit current command
C-x C-f		Open a file
C-b	Left	Move cursor left (b ackwards) one character
C-f	Right	Move cursor right (f orwards) one character
C-n	Down	Move cursor down one row (to n ext row)
C-p	Up	Move cursor up one row (to p revious row)
C-a	Home	Move cursor to start of line
C-e	End	Move cursor to e nd of line
C-x C-s		Save the file
M-<		Move to start of buffer
M->		Move to end of buffer
C-v	PageDown	Move one screenful down the buffer
M-v	PageUp	Move one screenful up the buffer
M-b		Move cursor b ackwards one word
M-f		Move cursor f orwards one word
M-a		Move cursor backwards one sentence
M-e		Move cursor forwards one sentence
M-{		Move cursor forwards one paragraph
M-}		Move cursor backwards one paragraph
C-x b		Switch to most recent other b uffer
C-x C-b		Split window and show list of b uffers
C-x o ("oh")		Switch to o ther half window
C-x 1 ("one")		Return to displaying 1 buffer
C-s		Interactive search forwards
C-r		Interactive search backwards
M-%		Search (forwards) and replace
	sub-options: y	SPC Change this instance and move on to the next.
	n	Don't change this instance but move to the next.
	q	RETURN Don't change this and quit.
	!	Change this instance and all future instances without prompting.
M-x goto-line		Prompt for a line to move to and then move to it.
C-u N M-x goto-line		Move to line <i>N</i> .
C-l		Centre current line in window.
C-u N C-l		Move current line to line <i>N</i> in window (0 = top line)
C-u N x		Insert <i>N</i> many letters <i>x</i> .
C-SPC		Mark the start of the region to be copied/cut.
C-w		Cut the marked region to the cut buffer.
M-w		Copy the marked region to the cut buffer.
C-y		Paste ("yank") the cut buffer into the document.
C-x (Start defining a keyboard macro.
C-x)		End of definition of keyboard macro.
C-x e		Execute keyboard macro.
C-u N C-x e		Execute keyboard macro <i>N</i> times.

Further reading

There is no end of literature on Emacs. The following is a short list of useful resources.

“Learning GNU Emacs”: Debra Cameron and Bill Rosenblatt, O'Reilly and Associates, Inc.

“Emacs Beginner's HOWTO”: Jeremy D. Zawodny,
<http://jeremy.zawodny.com/emacs/emacs.html>

“The Emacs Tutorial”: Paul Humke,
<http://www.stolaf.edu/people/humke/UNIX/emacs-tutorial.html>

“GNU Emacs Manual”: Free Software Foundation,
<http://www.gnu.org/software/emacs/manual/emacs.html>

“GNU Emacs Manual”: James Thornton,
<http://jamesthornton.com/emacs/node/emacs.html>