

# emacs/xemacs

DR. YIHSIANG LIOW (SEPTEMBER 17, 2024)

## Contents

1	What is emacs/xemacs?	2
2	Starting and exiting xemacs	4
3	File operations	7
4	Editing $\text{\LaTeX}$ files (ADDED 2024)	9
5	Clearing a command	10
6	Moving around a document	11
7	Configuring emacs for C/C++ source editing	13
8	Undo	14
9	Search and replace	15
10	Selecting, copying, pasting, and deleting a region	17
11	Indentation	21
12	Comments	24
13	Multiple frames	26
14	Compiling in emacs/xemacs	29
15	Shell	31
16	What's next?	32
17	Summary	33

**18 Summary: cursor movement****35**

# 1 What is emacs/xemacs?

*Emacs is a class of feature-rich text editors, usually characterized by their extensibility. Emacs has, perhaps, more editing commands compared to other editors, numbering over 1,000 commands. It also allows the user to combine these commands into macros to automate work.*

*Development began in the mid-70s and continues actively as of 2009. Emacs text editors are most popular with technically proficient computer users and computer programmers. The most popular version of Emacs is GNU Emacs, a part of the GNU project, which is commonly referred to simply as “Emacs”. The GNU Emacs manual describes it as “the extensible, customizable, self-documenting, real-time display editor.” It is also the most ported of the implementations of Emacs. As of July 2009, the latest stable release of GNU Emacs is version 23.1.*

*Aside from GNU Emacs, another version of Emacs in common use, XEmacs, forked from GNU Emacs in 1991. XEmacs has remained mostly compatible and continues to use the same extension language, Emacs Lisp, as GNU Emacs. Large parts of GNU Emacs and XEmacs are written in Emacs Lisp, so the extensibility of Emacs’ features is deep.*

*The original EMACS consisted of a set of Editor MACroS for the TECO editor. It was written in 1976 by Richard Stallman, initially together with Guy L. Steele, Jr. It was inspired by the ideas of TECMAC and TMACS, a pair of TECO-macro editors written by Steele, Dave Moon, Richard Greenblatt, Charles Frankston, and others.*

*In Unix culture, Emacs became one of the two main contenders in the traditional editor wars, the other being vi. The word “emacs” is often pluralized as emacsen, by analogy with boxen (itself used by analogy with oxen) and VAXen.*

– from wikipedia.org

*XEmacs is a graphical- and console-based text editor which runs on almost any Unix-like operating system as well as Microsoft Windows. XEmacs is a fork, developed based on a version of GNU Emacs from the late 1980s. Any user can download, use, and modify XEmacs as free software available under the GNU General Public License version 2 or any later version.*

– from wikipedia.org

For this tutorial, I assume you have access to Unix. So one of the following is true:

- You have a user account at `gandalf.ccis.edu` or `bilbo.ccis.edu`. You have read the PuTTY tutorial and know how to use PuTTY (or some other SSH client) to login to your account at `gandalf.ccis.edu` or `bilbo.ccis.edu`.
- You have access to a Linux machine (real or virtual).
- You have access to Cygwin on a Windows machine (this is not really Unix but a Unix emulation.)

Everything you learn in this tutorial applies to any Unix system with emacs/xemacs installed.

Emacs and xemacs are two very powerful text editor. I will only talk about some of the most commonly used operations.

Many of the operations uses special key bindings. These are special commands that you issue to emacs or xemacs from the keyboard. In fact you don't have to use the mouse at all. When you see this:

`C-x-s`

or

`C-x C-s`

it means “hold the [ctrl] key down, press and release `x`, and press and release `s`” (after that you may release the [ctrl] key). That's a command that you issue to save your file in emacs/xemacs. If you see something this

`M-g`

It means “hold the [alt] key and press and release the `g` key” (after that you may release the [alt] key.) You can also use the [esc] key. A quick warning: if you see something like

`C-x 1`

it means do “`C-x`” and then press `1`. So you do not hold the [ctrl] key down when you press `1`.

## 2 Starting and exiting xemacs

NEW 2024: For f40 (Fedora 40 virtual machine), in the bash shell, enter `su`, enter the root password, then execute

```
[student@localhost ~]$ update-vm
```

After this is done, close the bash shell. Open a new bash shell. To run emacs, instead of using `emacs`, execute `e`. For instance

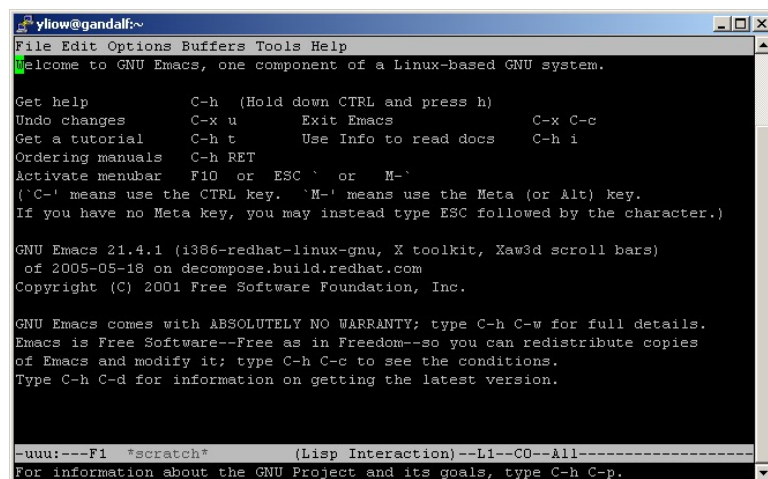
```
[student@localhost ~]$ e helloworld.cpp &
```

So in the rest of this pdf, remember to replace `emacs` with `e`.

To run emacs, go ahead and type this into your shell:

```
[student@localhost ~]$ emacs
```

You will see this:



```

yliow@gandalf:~
File Edit Options Buffers Tools Help
Welcome to GNU Emacs, one component of a Linux-based GNU system.

Get help          C-h (Hold down CTRL and press h)
Undo changes      C-x u          Exit Emacs          C-x C-c
Get a tutorial    C-h t          Use Info to read docs C-h i
Ordering manuals  C-h RET
Activate menubar  F10 or ESC ` or M-`
('C-' means use the CTRL key. 'M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)

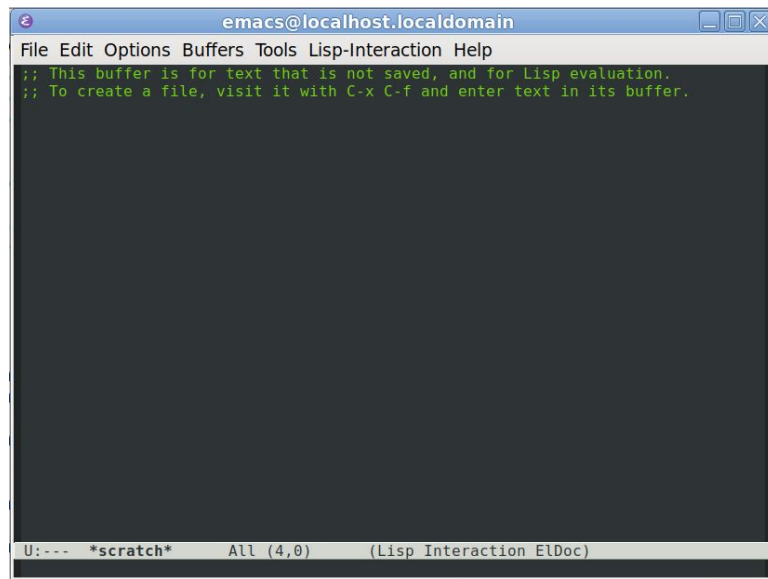
GNU Emacs 21.4.1 (i386-redhat-linux-gnu, X toolkit, Xaw3d scroll bars)
of 2005-05-18 on decompose.build.redhat.com
Copyright (C) 2001 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-d for information on getting the latest version.

--uuu:---F1 *scratch* (Lisp Interaction)--L1--C0--All-----
For information about the GNU Project and its goals, type C-h C-p.

```

or



(What you see as the initial/welcome screen depends on the version and the configuration.) This runs the emacs program. To run xemacs, do this instead:

```
[student@localhost ~]$ xemacs
```

Emacs and xemacs are very similar.

To exit emacs, in emacs do

`C-x-c`

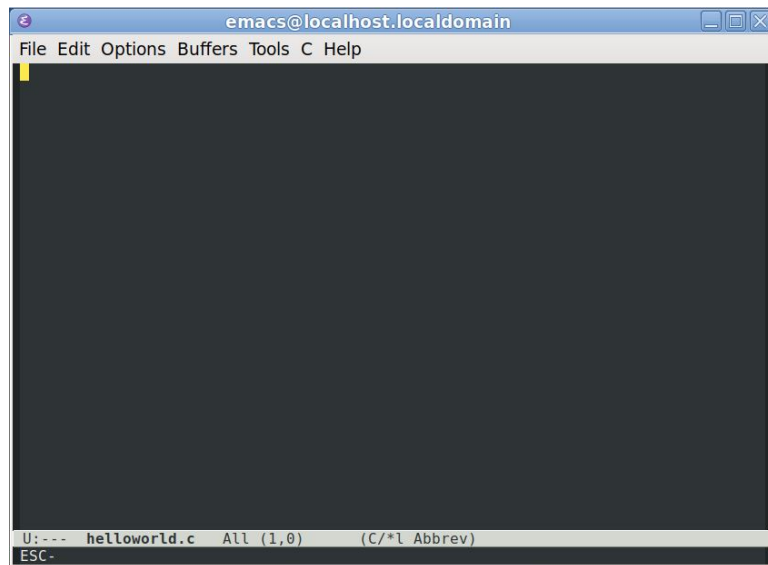
This means “hold the [ctrl] key down, press and release the `x` key, and press and release the `c` key”. You can release the [ctrl] key after that.

Please use one hand (your left) to do that!

**Exercise 2.1.** Start and exit emacs ten times. You have 10 seconds. □

Another way to start emacs is to open a file while you start the program. Try this in your bash shell:

```
[student@localhost ~]$ emacs helloworld.c
```



This will start emacs by loading the file `helloworld.c`. If `helloworld.c` exists, that file is loaded; otherwise a new file, `helloworld.c`, is created.

If you are using linux with GUI you should do

```
[student@localhost ~]$ emacs helloworld.c &
```

to run emacs in the background. This will make sure that your bash shell still respond to you.

### 3 File operations

Type this into your `helloworld.c`:

```
#include <stdio.h>

int main()
{
    printf("hello world\n");
    return 0;
}
```

To save the file do this:

`C-x-s`

Now exit emacs (recall: `C-x-c`) and you're back to your shell. At your shell's prompt view the contents of `helloworld.c` to verify that it's saved correctly execute the following in your shell:

```
[student@localhost ~]$ less helloworld.c
```

The program `less` will show you the contents of your file in your bash shell. To stop `less`, enter `q`.

To compile your `helloworld.c`, do

```
[student@localhost ~]$ gcc helloworld.c
```

This produces the binary executable `a.out`. To run `a.out`, you do

```
[student@localhost ~]$ ./a.out
```

and you'll see `hello world` appear in your bash shell:

```
[student@localhost ~]$ gcc helloworld.c
[student@localhost ~]$ ./a.out
hello world
[student@localhost ~]$
```

Note: to compile a C program, you use `gcc` and to compile a C++ program, you use `g++`. Note that the C language is a subset of the C++ language. Therefore `g++` can be used to compile both C and C++ programs.

**Exercise 3.1.** Remove `helloworld.c` (use `rm`). Recreate the file all over again, save and exit, remove the file. Do this five times altogether. You have 2 minutes. ☐



**Exercise 3.2.** Take a simple assignment from CISS240 (or just make up one yourself) and do the above activity five times or until you can remember how to start emacs, save a file, and exit emacs. ☐

If you want to open another file while you're already in emacs, you do

`C-x-f`

At this point emacs will prompt you for the filename. In fact you can specify the full path and enter the filename and press enter. Using the [tab] key emacs will perform path completion for you. You can also enter . (a period) at the bottom and emacs will give you a directory listing from which you can select a file to be loaded.

**Exercise 3.3.** Open emacs by loading a new file `blah.txt`. While in emacs, load `helloworld.c`. Exit emacs. ☐

If you want to insert a file into the file you're working at the point of your cursor you do this:

`C-x-i`

**Exercise 3.4.** Load emacs, creating a new file `helloworld2.c`. Type some text. Now insert `helloworld.c` into `helloworld2.c` at any point in your text. Save your `helloworld2.c` and exit emacs. View the contents of `helloworld2.c` using `less`. ☐

## 4 Editing $\LaTeX$ files (ADDED 2024)

When you are editing  $\LaTeX$  files (files ending with `.tex`), to get a double-quote, you need to press the double-quote key twice.

## 5 Clearing a command

Before we go further, I should let you know that sometimes you might be in the middle of a sequence of commands and you want to cancel the commands. The easiest way to do that is to press the [esc] key several time.

Let's try this. Recall that `C-x-f` load a file. Suppose you do this:

`C-x-f`

The bottom status bar of emacs now ask you for a filename. At this point, press the [esc] key a couple of times and you will see Quit. This will end the command to load a file.

Easy right?

## 6 Moving around a document

Besides moving around the document with arrow keys, page up and down, there are several other things that you will find useful.

For a programmer, one of the most common action is to go to a particular line based on its line number. Open emacs (in the background) with your `helloworld.c` and change it to this:

```
#include <stdio.h>

int main()
{
    printf("hello world\n")
    return 0;
}
```

(i.e. remove the semicolon for the first statement.) Save your file. Compile the program in your shell with

```
[student@localhost ~]$ gcc helloworld.c
```

and you should get an error:

```
helloworld.c: In function `main':
helloworld.c:6: error: parse error before "return"
```

This tells you that the C compiler encounters an error at line 6 before the keyword `return`. So now you want to open your program, go to line 6, and see if you can spot an error before the `return`.

To go to line 6 by doing this:

M-g-g

(i.e. hold the [alt] key, press and release `g` twice) you will be prompted for a line number. Enter 6 (and pressing the [enter] key) and your cursor will go to line 6.

Get it?

**Exercise 6.1.** Correct the error in `helloworld.c`, save the file, exit emacs, compile to verify that the error is corrected. Repeat the whole process (creating an error somewhere, compiling to get the line number close to the error, go to the line, correct the error, save the file, exit emacs, compile the program) 5 times. □

Instead of moving one character at a time using the arrow keys, you can also move word-wise. To move right by one word you do this:

C-[rightarrow]

To move left by one word do this:

C-[leftarrow]

If you have a long line you might want to move quickly to the end of the line. Do this:

C-e

(or use the [end] key) and to move to the front of the line you do this:

C-a

(or use the [home] key.)

**Exercise 6.2.** Open your `helloworld.c` and practice the following: Move to line 3, move right by one word, move to the beginning of the line, move to the end of line Now make up a few more of such exercises.

If your document is huge it might be convenient to also know how to get to the top and the bottom of the document. To go to the top do

M-<

and do this is you want to go to the bottom:

M->

**Exercise 6.3.** Obviously, you should now create some huge text documents and practice the above two commands.

## 7 Configuring emacs for C/C++ source editing

[If you're using my virtual machine, the emacs is already configured for programming C/C++. You may skip this section.]

The next section is on editing. Before we do that we want to configure your emacs for auto-indentation. I will not explain why it works since this is a configuration issue and to fully understand the configuration would require an understanding of emacs and the lisp programming language. Here's what you should do.

Go to your home directory. Then go into the `.emacs.d` directory. Open the file `init.el` with emacs. Add the following to the bottom of that file if it's not in the file:

```
(add-hook 'c-mode-hook 'turn-on-font-lock t)
(add-hook 'c++-mode-hook 'turn-on-font-lock t)
(setq auto-mode-alist (cons '("\\.C\\'" . c++-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.h\\'" . c++-mode) auto-mode-alist))
(setq c-default-style "bsd"
      c-basic-offset 4)
(line-number-mode 1)
(column-number-mode 1)
```

Save and exit.

That's it. Now your emacs understands C/C++ syntax coloring, indentation and the line number and column number is turned on.

You can configure emacs in many ways including changing the various syntax coloring and even the key bindings.

## 8 Undo

Probably the most important command is undo. Open a file in emacs, type in some random data and then do this:

`C-/`

or

`C-x u`

or

`C-_`

I usually do `C-/`.

## 9 Search and replace

Make a copy of your `helloworld.c` and call the new file `helloworld.cpp`. Open `helloworld.cpp` with `emacs`. Modify it to get this:

```
#include <stdio.h>

int main()
{
    printf("hello world\n");
    printf("hello galaxy\n");
    printf("hello columbian\n");
    return 0;
}
```

Let's do a search for the word `printf`. To forward search for a string you do this:

`C-s`

followed by the string you are searching for. To search forward to the next occurrence of the string you entered, you continually do `C-s`. Go ahead and do it.

To search backwards (reverse search), you pretty much do the same thing except that you use

`C-r`

instead of `C-s`. Go ahead and search backwards for `printf`.

Go to the line 1 of your file. Now let's make our program into a C++ program. We need to replace the `printf` with `cout`. First do this:

`M-%`

(You need the shift key to get to `%`.) `emacs` will prompt you for the string to replace. Enter

`printf(`

and then enter the string you want to replace `print(` with. Enter

`std::cout <<`

`emacs` will find each occurrence of `printf(` and ask you if you want to replace it with `std::cout <<`. You enter `y` (for yes) or `n` (for no) until `emacs` can't find anymore matches for `printf(`. Note that `emacs` search forward. Go ahead and change your program to this:



```
#include <stdio.h>

int main()
{
    std::cout << "hello world\n");
    std::cout << "hello galaxy\n");
    std::cout << "hello columbia\n");
    return 0;
}
```

**Exercise 9.1.** Using emacs replacement, get your program to look like this:

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    std::cout << "hello galaxy" << std::endl;
    std::cout << "hello columbia" << std::endl;
    return 0;
}
```

Save and exit. In your shell compile your program using g++ and run your program:

```
[student@localhost ~]$ g++ helloworld.cpp; a.out
```

**Exercise 9.2.** Open helloworld.cpp using emacs. Using search and replace modify the file so that it looks like this:

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl
               << "hello galaxy" << std::endl
               << "hello columbia" << std::endl;
    return 0;
}
```

Save and exit. In your shell compile your program using g++.



**Exercise 9.3.** Redo this whole section 3 times.



## 10 Selecting, copying, pasting, and deleting a region

To copy-and-paste a chunk of text, first you need to select the region to copy. You also need to do that if you want to delete a region. So let me first show you how to select a region ...

**Selecting a Region.** There are two ways to do this. First go to one end of the region. Now hold the shift key down and then use the arrow keys or pageup/pagedown key to go to the other end. You will see that the text you go over is highlighted – you’re basically selecting a region.

The above method does not work if you’re in a pure text-based environment for instance if you use some ssh program to login to your secret unix account in Russia. So another way to do this is to go to one end of the text you want to select and do

`C-spacebar`

and then go to the other end. In this type of environment, you might not see the highlighting.

Of course you can use your mouse to select a region too.

**Copying.** At the other end of the region you want to copy, do

`M-w`

Now emacs has a copy of the region that was marked.

**Paste a Region.** Now go to the place where you want to paste the region and do

`C-y`

This is called “yank” it in.

**Delete a Region.** To delete a region, again you have to select the region first. Once your region is selected, do

`C-w`

This is called “wipe” it out.

**Exercise 10.1.** Write a helloworld program:

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

Now using copy-and-paste to do this:

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    return 0;
}
```

Next change your program to this by removing a region:

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    return 0;
}
```

Redo this exercise 5 times.



**Exercise 10.2.** First write this C++ program:

```
#include <iostream>

int main()
{
    int x;
    std::cin >> x;

    return 0;
}
```

Change the above to this using select and copy:

```
#include <iostream>

int main()
{
    int x;
    std::cin >> x;
    int x;
    std::cin >> x;
    int x;
    std::cin >> x;

    return 0;
}
```

Change it to this using replace:

```
#include <iostream>

int main()
{
    int year;
    std::cin >> year;
    int x;
    std::cin >> x;
    int x;
    std::cin >> x;

    return 0;
}
```

Change it to this using replace:

```
#include <iostream>

int main()
{
    int year;
    std::cin >> year;
    int month;
    std::cin >> month;
    int x;
    std::cin >> x;

    return 0;
}
```

Finally change it to this using replace:

```
#include <iostream>

int main()
{
    int year;
    std::cin >> year;
    int month;
    std::cin >> month;
    int day;
    std::cin >> day;

    return 0;
}
```

## 11 Indentation

If your emacs is set up correctly you don't have to worry about indentation because emacs will do it for you automatically. There are however times when you need to re-indent. For instance, maybe you've accidentally inserted some extra spaces:

```
#include <stdio.h>

int main()
{
    std:: << "hello world\n" << std::endl;
    return 0;
}
```

All you need to do is to go to anywhere on the line to re-indent and press the tab key. Your cursor need not be at the beginning of the text.

There are also times when you need to re-indent a whole region. For instance suppose you have this chunk of code:

```
...
    int x;
    std::cin >> x;

    if (x > 0)
    {
        [line 1 of code]
        [line 2 of code]
        ...
        [line 50 of code]
    }
...
```

And then you realized ... duh! ... the code should be this:

```
...
    int x;
    std::cin >> x;

    [line 1 of code]
    [line 2 of code]
    ...
    [line 50 of code]
...
```

Now you need to indent all the 50 lines of code. You can use the tab key for each line to be indented. But the smarter thing to do is to indent a region instead of doing it one line at a time. So you first mark the region to indent (example: select the region using the shift key) and then do

C-M-\

That's it. Another way to do that is to do

M-x indent-region

You can use [tab] to complete emacs commands. Do this:

M-x inde[tab]-reg[tab]

**Exercise 11.1.** Write a dummy program like this:

```
int main()
{
    int x = 1;
    int x = 1;
    int x = 1;
    int x = 1;
    int x = 1;
}
```

(of course it won't compile ...) Now add these:

```
int main()
{
    {
        int x = 1;
        int x = 1;
        int x = 1;
        int x = 1;
        int x = 1;
    }
}
```

and perform an indentation of a region to get it to look like this:

```
int main()
{
    {
        int x = 1;
        int x = 1;
        int x = 1;
        int x = 1;
        int x = 1;
    }
}
```

Now make it go back to the original version. Do this 5 times. ☐

**Exercise 11.2.** Write the following file:

```
#include <iostream>
int main()
{
    std::cout << "hello world"
    << std::endl;
    return 0;
}
```

Select the whole file and indent it.





## 12 Comments

When writing C/C++ source files, you frequently need to comment and uncomment code. Try this.

In your C/C++ source file, select a region and then do

`C-c-c`

You will see that all your C/C++ code in that region is commented. Select that region again and do

`C-u-c-c`

and the lines in that region is now uncommented.

Get it?

**Exercise 12.1.** First write this:

```
if (x == 0)
a = 1;
b = 2;
c = 3;
if (y == 4)
d = 5;
e = 6;
f = 7;
```

Then modify the above to get this in the most efficient way:

```
if (x == 0)
{
    a = 1;
}
//b = 2;
//c = 3;
if (y == 4)
{
    d = 5;
    //e = 6;
    //f = 7;
}
```

and then this:

```
if (x == 0)
{
    a = 1;
    b = 2;
    c = 3;
    //if (y == 4)
    //{
    //    d = 5;
    //    //e = 6;
    //    //f = 7;
    //}
}
```

and then

```
if x == 0
{
    a = 1;
    b = 2;
    c = 3;
    if (y == 4)
    {
        d = 5;
        e = 6;
        f = 7;
    }
}
```

## 13 Multiple frames

When you write a program, you will frequently work with multiple files. Of course you can run multiple instances of emacs, running one instance of emacs for each file. But in fact you don't have to because a single instance of emacs can manage multiple files, each file in a buffer in emacs.

In fact it's a very bad idea to open multiple files with multiple instances of emacs. Why? Because you might accidentally have one file opened in two separate instances of emacs and you might be making *different* modifications to the same file. Of course when you save the program in one emacs instance, earlier program changes saved using the other emacs instance will be lost!!! See the problem?

So let me show you how to run one instance of emacs to manage any number of files.

First run this to write a hello world program:

```
[student@localhost ~]$ emacs helloworld.cpp &
```

Write your `helloworld.cpp` but don't exit yet. While in emacs do

`C-x 2`

You now have two frames: the original frame has split horizontally. Now do

`C-x o`

(Note: No `C` for the `o`, i.e., you do `C-x`, release the `Ctrl` key and press `o`.) This will move your cursor to the "other" window. You can also use

`C-tab`

Now do

`C-x 3`

and you current frame will split vertically. Do

`C-x 3`

a couple of times and you'll go from one frame to another. If you do

`C-x 1`

and your current frame will fill up the whole window – all other frames disappear. Create a few more frames again.

Now let's load a different file into a frame. Go to any frame and do

`C-x-f`

(see previous pages – I've already talked about `C-x-f` earlier.) At the bottom status bar, emacs will display your current path. You can enter a filename. If the file is not found, then emacs assumes you want create a new file. If the filename, emacs assumes you want to load the existing file into the frame. In any case you now have two files, each file occupying one buffer.

If you want to switch to a new buffer, you can list all buffers by doing this:

`C-x-b`

By the way, you can select a buffer by doing

`C-x b`

(Note: No `C` for the `b`).

When working with multiple frames, usually you want to save changes to the files in all frames. You can of course all through all the frame and save the files one at a time and do `C-x-s`. But you can also save all the files in all the frames like this: do `C-x s` (no `C` for the `s`) and then when prompted, you enter `!`.

**Exercise 13.1.** First write this program using emacs:

```
#include <stdio.h>

int main()
{
    printf("hello world\n");
    return 0;
}
```

Close your emacs. How open the file in two separate emacs instances:

```
[student@localhost ~]$ emacs helloworld.cpp &
[student@localhost ~]$ emacs helloworld.cpp &
```

In one emacs, modify the program:

```
#include <stdio.h>

int main()
{
    printf("top\n");
    printf("hello world\n");
    return 0;
}
```

Look at the other emacs and you'll see:

```
#include <stdio.h>

int main()
{
    printf("hello world\n");
    return 0;
}
```

The second emacs does not see the change you made in the first emacs. Close both emacs instances without saving the change. Now open the original program again using emacs. In emacs, split the frame vertically to get two frames. You'll see the same program in both frames. Modify the program in the left frame like this:

```
#include <stdio.h>

int main()
{
    printf("top\n");
    printf("hello world\n");
    return 0;
}
```

and you'll see the other frame catches the change. Modify the program in the right frame like this:

```
#include <stdio.h>

int main()
{
    printf("top\n");
    printf("hello world\n");
    printf("bottom\n");
    return 0;
}
```

and you'll see the other frame catches the change.

## 14 Compiling in emacs/xemacs

Another thing you do frequently is to compile your C/C++ source files into executables. Of course you can compile your source files in the shell. However you can compile in emacs. The benefit of compiling in emacs, is that emacs can point you directly to the location of the compilation error.

Enough talk ... let's do something.

First run emacs to write main.cpp like this:

```
#include <iostream>

int main()
{
    std::cout << "hello world" std::endl;
    return 0;
}
```

Note the error.

While in emacs do

M-x

at the bottom status bar of emacs, you will see

M-x

Enter `compile` so that the bottom status bar looks like

M-x `compile`

Press the enter key. The bottom status bar will become

Compile command: `make -k`

Change it to

Compile command: `g++ *.cpp`

and press the [enter] key. If you did not save your program, emacs will ask if you want to save it. Answer `y`.

Next, you will see emacs executing `g++` and in this case because there are program errors emacs will open another frame showing you `g++` error message(s). Point your cursor to the first error and press the [enter] key. Emacs will point your cursor to the error in your

source file. Correct the error so that your `main.cpp` looks like this:

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

Once again do

`M-x`

and modify the bottom status bar to

`M-x compile`

and press the [enter] key. The bottom status bar will show your compile command that you have entered. This time `g++` compiles your program successfully. Now you can go to your shell and run your program `a.out`.

Of course if you have a makefile, then for the compile command you can enter `make`.

Compile command: `make`

(See the tutorial on `make`.)

**Exercise 14.1.** Now write a C++ program that involves multiple files – opening emacs only once. For instance write a function `helloworld()` in `hw.cpp` that prints the hello world string and put the header in a header file `hw.h`. Finally write a `main.cpp` that calls `helloworld()`. Do this about 5 times. □

## 15 Shell

In the previous section I talked about compiling your program. But it gets even better ...

You can also execute your program `a.out` in emacs as well!!! In fact you can run a shell inside emacs and execute any linux command. So you can pretty much do everything inside emacs! To try it out, execute

```
M-x shell
```

and you will have a shell inside your emacs.

One thing to note: In your usual terminal bash shell, you can use the arrow keys to get the previous and next command in the bash history. However inside emacs when you're running a shell, the arrow keys move your cursor around in the shell output. So instead of using arrow keys, you use `M-p` for the previous shell command and `M-n` for the next shell command.

**Exercise 15.1.** Use emacs to write a C/C++ program. Make sure there are at least 3 errors in your program. Compile your program within emacs and correct the errors one at a time, re-compiling after each correction. Run your program within emacs. Do the above 5 times.



## 16 What's next?

This is only a short intro to emacs. There are lots of emacs tutorials on the web. You can find out more using google. Another to do is to google for the “emacs refcard” (reference card) which contains a summary of emacs commands. Of course there's also the official emacs web site.

## 17 Summary

C-x-c	Exit emacs/xemacs
C-x-s	Save file
C-x-f	Load file in current frame
C-x i	Insert another file into current file
M-g M-g	Go to line number (M-g for xemacs)
C-[rightarrow]	Go to next word
C-[leftarrow]	Go to previous word
C-a	Go to beginning of line
C-e	Go to end of line
M-<	Go to top
M->	Go to bottom
C-/	Undo
C-x u	Undo
C-_	Undo
C-s	Search
C-r	Reverse search
M-%	Replace
C-spacebar	Mark beginning of region. (Or just hold the shift key.)
M-w	Copy region
C-y	Paste: "yank" it in
C-w	Delete region: "wipe" it out
[tab]	Indent a line
C-M-\	Indent region
C-c-c	Comment region
C-u-c-c	Uncomment region
M-;	Toggle comment/uncomment
C-x 1	Keep 1 frame
C-x 2	Split frame vertically
C-x 3	Split frame horizontally
C-x o	Go to other frames
C-tab	Go to other frames
C-x-b	Show all buffers

C-x b	Select buffer
M-x compile	Compile (and enter the compile command if necessary)
M-x shell	Shell

## 18 Summary: cursor movement

Without arrow keys:

	CHARACTER	WORD	LINE	PARAGRAPH	PAGE	FILE	LINE-NUM
right/forward	C-f	M-f	C-e	M-}	C-v	M-<	M-g M-g
left/backward	C-b	M-b	C-a	M-{	M-v	M->	
up/previous	C-p						
down/next	C-n						

Using arrow keys:

	CHARACTER	WORD
right/forward	RIGHT	C-RIGHT or M-RIGHT
left/backward	LEFT	C-LEFT or M-LEFT
up/previous	UP	
down/next	DOWN	