KAIST
EE209: Programming Structures for EE
A Minimal EE 209 Computing Environment

1. Your Account in LabMachine

One time only…

Notes:

• LabMachine is a cluster of computers that is administered by TA.

• The LabMachine consists of 13 (143.248.141.52~143.248.141.64) computers which have Fedora 12.

• The local computer communicates with LabMachine via a terminal emulation program that can use the SSH protocol. Two such programs are PuTTY (for MS Windows) and Terminal (for Mac OS X).

1.1. Your student id number is **your account ID**.

1.2 Password of your account ID was mentioned in your ee209 lecture. All students have the same password. Make sure to change the password on **all** 13 (143.248.141.52~143.248.141.64) machines.

> Password Change Command is "passwd"
> If you issue the command "passwd",
> In response to the "(current) UNIX password: ", type current password
> In response to the "New password: ", type your new password.
> In response to the "Retype new password: ", type your new password again.
> Note: The change of password is limited to the computer you logged in. You have to change the password on all 13 machines.

1.3 Each machine has an independent file system; i.e. if you create/update a file in one machine, that change will not appear in any other sibling machine. This also means that in case a machine crashes or goes offline (although that rarely occurs), you will potentially lose your files from the system. We recommend you to regularly back up all program files in your local machine. You should preferably develop your programs in local machines and use the lab machines for testing.

2. LabMachine Terminal Session

2.1. Using a Lab Computer Running Microsoft Windows

2.1.1. Launch PuTTY[1].
From the "Start | All Programs | PuTTY" menu, click on PuTTY.

2.1.2. Log into LabMachine.
In PuTTY:
Click on the "Window | Colours" Category, and make sure the "Use system colours" checkbox is checked. Click on the "Session" Category. In the "Host Name (or IP address)" text box, type any LabMachine IP (143.248.141.52 ~ 143.248.141.64). Make sure that the "Port" text box contains "22". Make sure the "Connection type" radio button panel is set to "SSH". Make sure the "Close window on exit" radio button panel is set to "Only on clean exit". Click on the "Open" button.

In the resulting PuTTY window:
If you log into LabMachine for the first time, you will see a "PuTTY Security Alert" warning message. If you click "Yes", this message is never shown again. In response to the "login as:" prompt, type your user id followed by the Enter key. In response to the "password:" prompt, type your password followed by the Enter key. (The password will not echo as you type.) A successful log-in will show you a Unix shell prompt.

2.1.3. Log out of LabMachine.
In PuTTY, issue the "logout" (or "exit") command to disconnect the client from LabMachine (PuTTY will exit automatically).

2.2. Using a Lab Computer Running Mac OS X:

2.2.1. Open a Terminal window.
Click on the "Terminal" button at the bottom of the screen; its icon is a video display with a cursor.

2.2.2. Log into LabMachine.
In the terminal window:
Issue the command "ssh *yourUserId*@143.248.141.52"; the available IPs are: 143.248.141.52 ~ 143.248.141.64
If an SSH-related message appears, type "Yes". Type your password, followed by the Enter key.

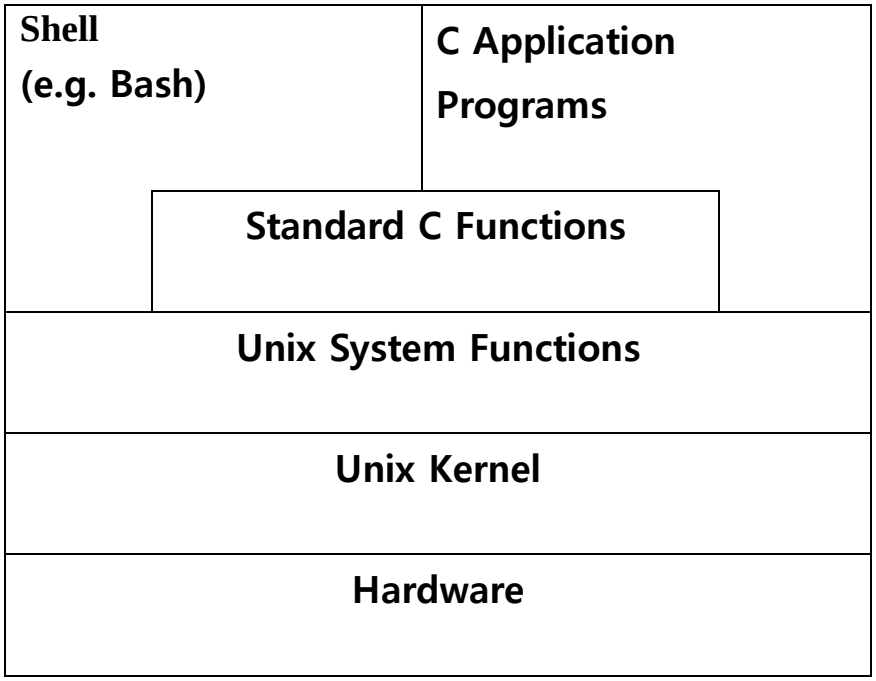
2.2.3. Log out of LabMachine.
In the terminal window:
Issue the "exit" or "logout" command.

2.2.4. Close the Terminal window.
Issue the "exit" or "logout" command.

[1]If you want to install Putty, use a web browser to visit the page http://www.putty.org/. Click on the "You can download PuTTY here" anchor. On the new page, click on the "putty.exe" anchor. In the "File Downloading" dialog box, click on the "Save" button. In the "Save As" dialog box, choose some appropriate location in your local file system. Then launch PuTTY by double-clicking on the putty.exe file via Windows Explorer.

KAIST

EE 209: Programming Structures for EE
Unix and Bash



KAIST
EE 209: Programming Structures for EE
Unix and Bash

Filenames and Directorynames	
/dir1/.../dirN	Absolute dname
dir1/.../dirN	Relative dname
/dir1/.../file	Absolute fname
dir1/.../file	Relative fname

Special Filename and Directoryname Characters	
*fnameord*name*	* matches 0 or more characters
fnameord?name*	? matches any single character
"*fname or dname*"	" allows whitespace in a dname or fname
'*fname or dname*'	' allows whitespace in a dname or fname
fnameord\'*name*	Backslash (escape) character allows special characters in a dname or fname
~loginid	Home directory of *loginid*
~	Your home directory
..	Parent of working directory
.	Working directory

Special Command Characters	
command 0< *fname* *command* < *fname*	Redirect stdin to *fname*
command 1> *fname* *command* > *fname*	Redirect stdout to *fname*
command 2> *fname*	Redirect stderr to *fname*
command 1> *fname* 2>&1	Redirect stdout and stderr to *fname*
command1 | *command2*	Pipe from *command1* to *command2*
^d	End of file
command &	Run *command* as a background process
^z	Turn my foreground process into a stopped background process
^c	Send a SIGINT signal
↑	Scroll backward through the command history list
↓	Scroll forward through the command history list
!*prefix*	Reissue the most recently issued command that begins with *prefix*
!*commandnum*	Reissue the command whose number is *commandnum* (see the “history” command)

Commands

Commands marked with "(Bash)" are shell built-in commands. Commands marked with "(bin)" are executable binary files.

Command for Getting Help	
man [*section*] *pagename*	(bin) Print to stdout the Unix manual page (from *section*) whose name is *pagename*. Section 1 describes commands and utilities (e.g. cat, ls). Section 2 describes Unix system calls (e.g. fork, dup). Section 3 describes library functions (e.g. printf(), strlen()).

Configuration Commands	
source *fname*	(Bash) Execute the shell script in *fname*
export *variable*=*value*	(Bash) Set environment *variable* to *value*
export PATH=*dname1*:*dname2*:...	(Bash) Set the PATH environment variable indicating that Bash should search *dname1*, *dname2*, ... to find commands that are specified as relative fnames
export MANPATH=*dname1*:*dname2*:...	(Bash) Set the MANPATH environment variable indicating that the man command should search *dname1*, *dname2*, ... to find man pages
variable=*value*	(Bash) Set shell *variable* to *value*
PS1="\h:\w\$ "	(Bash) Set the PS1 shell variable to indicate that the command prompt should contain the name of the host computer, a colon, the name of the working directory, a dollar sign, and a space
set –o *shelloption*	(Bash) Turn on *shelloption*
set +o *shelloption*	(Bash) Turn off *shelloption*
set –o ignoreeof	(Bash) Turn on the ignoreeof shell option to indicate that ^D entered at the Bash prompt should not terminate Bash
set –o noclobber	(Bash) Turn on the noclobber shell option to indicate that Bash should not overwrite files via redirection
alias *aliasname*=*string*	(Bash) Create an alias definition such that *aliasname* as an abbreviation for *string*
unalias *aliasname*	(Bash) Destroy the alias definition that defines *aliasname*

Directory-Related Commands	
pwd	(Bash, bin) Print the name of the working directory to stdout
cd [*dname*]	(Bash) Make *dname* the working directory
ls [-la] [*dname*]	(bin) List the contents of *dname* to stdout
ls [-la] [*fname*]	(bin) List the attributes of *fname* to stdout
mkdir *dname*	(bin) Create *dname*
rmdir *dname*	(bin) Destroy the empty directory *dname*

File-Related Commands	
cat	(bin) Concatenate (print) stdin to stdout
cat *fname* ...	(bin) Concatenate (print) *fname* ... to stdout
more *fname* ...	(bin) Print *fname* ... to stdout one screen at a time
less *fname* ...	(bin) Print *fname*, ... to stdout one screen at a time The man command pipes its output through less
xxd *fname*	(bin) Hexdecimal dump *fname* to stdout
cp [-i] *sourcefname targetfname*	(bin) Copy *sourcefname* to *targetfname*
cp [-i] *sourcefname targetdname*	(bin) Copy *sourcefname* to *targetdname*
cp –r *sourcedname targetdname*	(bin) Copy (recursively) *sourcedname* to *targetdname*

mv [-i] *sourcefname targetfname*	(bin) Rename *sourcefname* to *targetfname*
mv [-i] *sourcefname ... targetdname*	(bin) Move *sourcefname* ... to *targetdname*
rm [-i] *fname* ...	(bin) Remove *fname* ...
rm –r [-i] *dname* [*fname* ...]	(bin) Remove *dname* (recursively) and *fname* ...

File and Directory Permission Commands	
chmod *mask fnameordname* ...	(bin) Set the permissions of *fnameordname* ... as indicated by *mask*
umask *mask*	(Bash) Set the default permissions used when creating new files and directories as indicated by *mask*

Software Development Commands	
emacs	(bin) Create or edit a text file using the Emacs editor
gcc209	(bin) Preprocess, compile, assemble, and link a program using options appropriate for EE 209; a variant of gcc
gdb	(bin) Debug a program
make	(bin) Build a program
ar	(bin) Create an archive file containing object code
gprof	(bin) Analyze the performance of a program

Miscellaneous Commands	
history	(Bash) Print a numbered command history list to stdout
passwd *oldpassword*	(bin) Change my password from *oldpassword*
wc [*fname* ...]	(bin) Print a count of characters, words, and lines in *fname* ... (or stdin) to stdout
date	(bin) Print the date and time to stdout
printenv [*variable*]	(bin) Print the definition of environment *variable* (or of all environment variables) to stdout
echo [*arg* ...]	(Bash, bin) Print *arg* ... to stdout
who	(bin) Print information about current users to stdout
grep *pattern fname* ...	(bin) Print each line of *fname* that contains *pattern* to stdout
sort [*fname*]	(bin) Print each line of *fname* (or stdin) in lexicographic order to stdout
diff *fname1 fname2*	(bin) Print an indication of the differences between the contents of *fname1* and *fname2* to stdout
which *command*	(bin) Search PATH for *command*, and print the dname where it was found to stdout

Process Control Commands	
jobs	(Bash) List the names and jobnums of my background processes to stdout
fg [%*jobnum*]	(Bash) Move my background process with the given *jobnum* to the foreground
bg [%*jobnum*]	(Bash) Turn my stopped background process into a running background process
kill [–*signal*] %*jobnum*	(Bash) Send *signal* to my background process with the given *jobnum*
ps	(bin) Display a list of my processes
kill [–*signal*] *pid*	(bin) Send *signal* to the process whose id is *pid*
exit	(Bash) Exit Bash
logout	(Bash) Exit Bash and the terminal session

KAIST
EE 209: Programming Structures for EE
Emacs Tutorial

This tutorial describes how to use a minimal subset of the Emacs editor. See the Emacs summary sheet distributed in precept for more information. Also see Chapter 3 of our *Programming with GNU Software* (Loukides & Oram) textbook, and http://www.gnu.org/software/emacs/.

The tutorial assumes that you have copied the necessary files from http://www.ndsl.kaist.edu/~kyoungsoo/ee209/precepts/01/src/. Specifically, we assume you have copied files named hello.c and circle.c into your working directory. You can download them using *wget:*

`wget http://www.ndsl.kaist.edu/~kyoungsoo/ee209/precepts/01/src/`*`filename`*

to your working directory.

Throughout the tutorial text in **boldface** indicates hands-on activities.

Background

Emacs was created in the mid-1970s by Richard Stallman. Originally it was a set of "*e*diting *mac*ros" for an editor that now is extinct. Emacs is popular, for a few reasons. Emacs is free and it is a component of the GNU tool set from the Free Software Foundation. It is highly customizable: Emacs is written in the LISP programming language, and is easy to configure via that language. Emacs is integrated with other GNU software. In particular, Emacs is integrated with the Bash history mechanism. Essentially you can think of the Bash history list as a "file"; you can use Emacs commands to scroll through and edit that file, and thereby easily reissue previous commands or variants thereof. Emacs also is integrated with the GCC compiler driver, as this tutorial describes. Finally, and probably most importantly, Emacs is integrated with GDB debugger. A future precept will describe that integration.

Emacs is a "modal" editor. That is, at any given time, Emacs is in one of several modes. In the EE 209 course you will use "C mode," "Assembler mode," and "Text mode." Emacs determines its mode based upon filename extensions. If the current file has a name whose extension is ".c", then Emacs will be in "C mode." If the current file has a name whose extension is ".s", then Emacs will be in "Assembler mode." By default, Emacs is in "Text mode."

Launching Emacs

To launch Emacs, issue the `emacs` command followed by the name of the file that you wish to create or edit. For example, **issue this command at the Bash prompt:**

`emacs circle.c`

Emacs loads the contents of the circle.c into a buffer in memory, and displays that buffer in the window. It places the point over the first character in the first line of the buffer.
Note the Emacs terminology: A *buffer* is an area of memory. A *window* is a graphical entity that displays the contents of a specified buffer. The *point* is a small black box which overlays a character, thus indicating which character is the "current" character.

Notation

Throughout this document:
• "`Esc` *`somechar`*" means "type the Esc key followed by the *somechar* key."

• "`Ctrl-`*`somechar`*" means "type the *somechar* key while holding down the Ctrl key."

for any character *somechar*.
Incidentally, "`Alt-`*`somechar`*" (that is, type the *somechar* key while holding down the Alt key) has the same effect in Emacs as "`ESC` *`somechar`*" does.

Calling Functions

In Emacs, all work is accomplished by calling functions. The syntax for calling a function is:
`Esc x` *`function`*

For example, the `forward-char` function moves the point forward one character:
`Esc x forward-char`

Emacs moves the point forward one character within the buffer each time you call the `forward-char` function. **Call `forward-char` a few times.**
Clearly there must be a better way to move the point! More generally, there must be a better way to call often-used functions.

Key Bindings

There indeed is a better way. The most often-used functions are bound to keystrokes.
For example, the `forward-char` function is bound to the keystroke `Ctrl-f`. **Type `Ctrl-f` a few times.** The `forward-char` function also is bound to the right-arrow key. **Type the right-arrow key a few times.**

Many keystrokes are bound by default. You also can bind your own, typically by placing a function call of this form in your .emacs file:
`(global-set-key` *`keystrokes`* `'`*`function`*`)`

But few new Emacs users create their own keystroke bindings.

Moving the Point

The simplest way to move the point is via the `forward-char`, `backward-char`, `next-line` and `previous-line` functions, each of which is bound to an arrow key. **Type the arrow keys to move the point right, left, down, and up several times.**

The `beginning-of-line` and `end-of-line` functions have intuitive meanings. They are bound to the `Ctrl-a` and `Ctrl-e` keystrokes, respectively. They may also be bound to the `Home` and `End` keys, respectively; but `Home` and `End` may or may not work with your terminal emulation software. **Type `Ctrl-a`, `Ctrl-e`, `Home`, and `End` several times.**

Perhaps counter-intuitively, the `scroll-up` function moves the window downward in the buffer; equivalently, it moves the buffer upward in the window. The `scroll-up` function is bound to `Ctrl-v`, and also may be bound to the `PageDn` key. The `scroll-down` function moves the window upward in the buffer. That is, it moves the buffer downward in the window. The `scroll-down` function is bound to `ESC v`, and also may be bound to the `PageUp` key. **Type `Ctrl-v`, `PageDn`, `ESC v`, and `PageUp` several times.**

The `end-of-buffer` function moves the point to the end of the buffer; it is bound to `Esc >`. The `beginning-of-buffer` function moves the point to the beginning of the buffer; it is bound to the `Esc <`. **Type `Esc >` and `Esc <` several times.**

The `goto-line` function allows you to specify, by number, the line to which the point should be moved. It is bound to the `Ctrl-x l` (that's Ctrl-x followed by the "ell" key) keystroke sequence. **Type `Ctrl-x l`, followed by some reasonable line number, followed by the `Enter` key.**

Inserting and Deleting

To insert a character, move the point to the character before which the insertion should occur, and then type the character. **Move the point to some arbitrary spot in the buffer, and type some characters.**

The `c-electric-backspace` function (bound to the `Backspace` key) deletes the character before the point. **Move the point to some arbitrary spot in the buffer, and type `Backspace` several times.** The `c-electric-delete-forward` function (bound to `Ctrl-d`) deletes the character at the point. **Move the point to some arbitrary spot in the buffer, and type `Ctrl-d` several times.**

To delete a line, move the point to the beginning of the line and then call the `kill-line` function (bound to `Ctrl-k`). Calling the function once kills the characters comprising the line, but not the line's end-of-line mark. Calling the function a second time also kills the end-of-line mark. **Move the point to the beginning of some arbitrary line, and type `Ctrl-k` several times.**

Actually, the `kill-line` function doesn't completely discard the line that it kills; instead it moves the line to the Emacs clipboard. The `yank` function (bound to `Ctrl-y`) copies ("yanks") the line from the Emacs clipboard into the buffer at the point. The combination of the

`kill-line` and `yank` functions provides a single-line cut-and-paste functionality, as this sequence illustrates:

- **Move the point to the beginning of some non-empty line that you wish to move.**
- **Type `Ctrl-k` twice.**
- **Move the point.**
- **Type `Ctrl-y`.**

For multiple-line cut-and-paste, you must know about Emacs *regions*. A region is an area of text that is bounded by the point and the *mark*. The `set-mark-command` function (bound to `Ctrl-Space`) sets the mark. The `kill-region` function (bound to `Ctrl-w`) moves the region to the Emacs clipboard; effectively it *w*ipes out the region. This sequence illustrates moving multiple contiguous lines from one place to another in the buffer:

- **Move the point to the beginning of the first line that you wish to move.**
- **Type `Ctrl-Space` to set the *mark*.**
- **Move the point to the end of the last line that you wish to move. Note that Emacs highlights the *region* thus bounded by the point and the mark.**
- **Type `Ctrl-w` to "wipeout" the region. Emacs moves the region to its clipboard.**
- **Move the point to some spot in the buffer**
- **Type Ctrl-y to yank (that is, copy) the text from the clipboard to the buffer at the point.**

(Note that the "minimal computing environment" described in our first precept is completely mouseless. To use the mouse, you can install an X Window System Server on your computer.)

Saving and Exiting

The `save-buffer` function (bound to `Ctrl-x Ctrl-s`) saves the buffer, that is, copies the contents of the buffer to its file on disk. **Type `Ctrl-x Ctrl-s` to save the buffer to the circle.c file.** As its name implies, the `save-buffers-kill-emacs` function (bound to `Ctrl-x Ctrl-c`) saves all Emacs buffers to their respective files on disk, and exits Emacs. (The section of this tutorial entitled "Managing Windows and Buffers" describes how you can use more than one Emacs buffer simultaneously.) **Type `Ctrl-x Ctrl-c` to exit Emacs**, thus returning to the Bash prompt.

Indenting

At this point circle.c probably is seriously mangled. So **recopy the circle.c file from the above URLs to your working directory. Then issue the command `emacs circle.c` to relaunch Emacs to edit the circle.c file.**

Emacs automatically indents C code as you type it, according to the indentation style that you specified in your .emacs file.

The `c-indent-command` function (bound to the `Tab` key) indents the current line according to the chosen indentation style. Note that the Tab key does not insert a tab character into your file; rather it indents the current line. **Intentionally mal-indent a line, move the point to any spot within that line, and type the `Tab` key.**

The `indent-all` function (bound to `Ctrl-x p` because it indents your code *p*erfectly) indents all lines of the buffer according to the chosen indentation style. **Intentionally mal-indent multiple lines scattered throughout the buffer, and then type `Ctrl-x p`.**

Searching and Replacing

The `isearch-forward` function (bound to `Ctrl-s`) incrementally searches forward through the buffer for the text that you specify. This sequence illustrates:

- **Move the point to the beginning of the buffer.**
- **Type Ctrl-s, followed by the text "i1"**
- **Type Ctrl-s repeatedly.**
- **Move the point, thereby ending the search.**

The similar `isearch-backward` function (bound to `Ctrl-r`) incrementally searches backward through the buffer.

The `query-replace` function (bound to `Esc %`) incrementally replaces the "old" text that you specify with the "new" text that you specify. During execution of the function, typing "y" commands Emacs to perform the replacement and continue executing the function, "n" commands Emacs to skip the replacement and continue executing the function, "!" command Emacs to perform all replacements and stop executing the function, and "q" commands Emacs to stop (*q*uit) executing the function. For example:

- **Move the point to the beginning of the buffer.**
- **Type `Esc %`, followed by "i1", followed by "xxx".**
- **Type "y" and "n" a few times.**
- **Type "q".**
- **Move the point to the beginning of the buffer.**
- **Type `Esc %`, followed by "xxx", followed by "i1".**
- **Type "!".**

Managing Windows and Buffers

Recall that, in Emacs jargon, a *buffer* is a region of memory, and a *window* is a graphical area which displays the contents of a buffer. So far in this tutorial you've used only one buffer and one window. More generally, at any given time, Emacs will be managing multiple buffers and will be displaying some (but not necessarily all) of them in windows.

To "find" a file means to load it into a buffer. The `find-file` function (bound to `Ctrl-x Ctrl-f`) finds the file whose name you provide. **Type `Ctrl-x Ctrl-f hello.c` followed by the `Enter` key** to load the hello.c file into a buffer. Then **type `Ctrl-x Ctrl-f circle.c` followed by the Enter key** to load the circle.c file into a buffer. At this point Emacs is managing two buffers; one of them is displayed in a window.

The `split-window-vertically` function (bound to `Ctrl-x 2`) splits the current window into two windows, each of which displays the same buffer. **Type `Ctrl-x 2` to split the current window into two windows.** The `other-window` function (bound to `Ctrl-x o`) moves the point to the other window. **Type `Ctrl-x o` a few times** to move the point back-and-forth between the two windows. Now **type `Ctrl-x Ctrl-f hello.c`** to find the

hello.c file. At this point Emacs is managing two buffers; two of them are displayed in Emacs windows.

The `delete-other-window` function (bound to `Ctrl-x 1`) deletes the other window (that is, the window in which the point does not reside), thus returning Emacs to its default one-window state. **Type `Ctrl-x o` as necessary to move the point to the window that displays the hello.c buffer. Type `Ctrl-x 1`** to delete the window that displays the circle.c buffer, leaving only the window that displays the hello.c buffer. At this point Emacs is managing two buffers; only one of them – the hello.c buffer – is displayed in a window.
With today's windowing operating systems, the ability of Emacs to manage multiple windows is less important than it used to be. However, you must know about Emacs windows to (1) use GDB within Emacs, as will be described in an upcoming precept, and (2) build within Emacs, as described in the next section of this tutorial.

Building

Most EE 209 students build (that is, preprocess, compile, assemble, and link) C programs by issuing the `gcc209` command at the shell prompt. An alternative is to build C programs by issuing the `gcc209` command from within Emacs. The alternative approach is optional in the EE 209 course.
The `compile` function (no keystroke binding) builds a C program from within Emacs using whatever command you specify. This sequence illustrates:
• **Type `Esc x compile`.** Emacs assumes that you wish to use the "make –k" command to build. At this point in the course, that's incorrect. So **type the `Backspace` key repeatedly** to delete that command. **Then type: `gcc209 circle.c –o circle`.**

KAIST
EE 209: Programming Structures for EE

This reference sheet assumes that Emacs is configured using the .emacs file provided to EE 209 students.

To type "Ctrl-*somechar*" (for any character *somechar*), type the *somechar* key while holding down the Ctrl key. To type "Esc *somechar*" (for any character *somechar*), type the Esc key followed by the *somechar* key. Typing "Alt- *somechar* " has the same effect as typing "Esc *somechar*".

In Emacs all work is accomplished by calling functions. To call a function, type "Esc x *function*".

Many functions are bound to keystrokes.

Commonly used functions are in **boldace**.

Moving the Point

Binding	Function	Description
→	**forward-char**	**Move the point forward one character**
←	**backward-char**	**Move the point backward one character**
↓	**next-line**	**Move the point to the next line**
↑	**previous-line**	**Move the point to the previous line**
Ctrl-f	forward-char	Move the point forward one character
Ctrl-b	backward-char	Move the point backward one character
Ctrl-n	next-line	Move the point to next line
Ctrl-p	previous-line	Move the point to previous line
Esc f	forward-word	Move the point to next word
Esc b	backward-word	Move the point to previous word
Home	**beginning-of-line**	**Move the point to beginning of line (but not with some terminal apps)**
End	**end-of-line**	**Move the point to end of line (but not with some terminal apps)**
Ctrl-a	**beginning-of-line**	**Move the point to beginning of line**
Ctrl-e	**end-of-line**	**Move the point to end of line**
Esc a	c-beginning-of-statement	Move the point to the beginning of C statement
Esc e	c-end-of-statement	Move the point to the end of C statement
PageDn	**scroll-up**	**Move the point to next page (but not with some terminal apps)**
PageUp	**scroll-down**	**Move the point to previous page (but not with some terminal apps)**
Ctrl-v	**scroll-up**	**Move the point to next page**
Esc v	**scroll-down**	**Move the point to previous page**
Esc <	**beginning-of-buffer**	**Move the point to beginning of the buffer**
Esc >	**end-of-buffer**	**Move the point to end of the buffer**
Esc Ctrl-a	beginning-of-defun	Move the point to beginning of the C function
Esc Ctrl-e	end-of-defun	Move the point to end of the C function
Ctrl-x l *line*	**goto-line**	**Move the point to line whose number is *line***

Inserting and Deleting

Binding	Function	Description
Bsp	**c-electric-backspace**	**Delete the character before the point**
Esc Bsp	backward-kill-word	Delete the characters from the point to the beginning of the word
Ctrl-d	**c-electric-delete-forward**	**Delete the character at the point**
Ctrl-k	**kill-line**	**Cut the current line**
Ctrl-Sp	**set-mark-command**	**Set the mark at the point**
Ctrl-x Ctrl-x	exchange-point-and-mark	Exchange the mark and the point
Ctrl-x h	mark-whole-buffer	Set the point at the beginning and the mark at the end of the buffer
Ctrl-w	**kill-region**	**Cut the region denoted by the mark and the point**
Esc w	kill-ring-save	Copy the region denoted by the mark and the point
Ctrl-y	**yank**	**Paste the previously cut/copied region at the point**

Saving and Exiting

Binding	Function	Description
Ctrl-x Ctrl-s	**save-buffer**	**Save the current buffer to its file**
Ctrl-x Ctrl-w *file*	write-file	Write the current buffer to *file*
Ctrl-x Ctrl-q	vc-toggle-read-only	Toggle the current buffer between read-only and read/write
Ctrl-x Ctrl-c	**save-buffers-kill-emacs**	**Save all buffers and exit Emacs**

Indenting

Binding	Function	Description
Ctrl-c .	c-set-style	Set the C indentation style to the specified one
TAB	**c-indent-command**	**Indent the current line of the C program**
Esc Ctrl-\	indent-region	Indent the region of the C program denoted by the mark and the point
Ctrl-x p	**indent-all**	**Indent all lines of the C program (i.e. indent the program perfectly)**

Searching and Replacing

Binding	Function	Description
Ctrl-s *string*	**isearch-forward**	**Search forward for *string***
Ctrl-r *string*	isearch-backward	Search backward for *string*
Esc % *old new*	**query-replace**	**Replace the *old* string with the *new* one** **y => replace** **n => skip** **! => replace all** **q => quit**

Managing Windows and Buffers

Binding	Function	Description
Ctrl-x Ctrl-f *file*	**find-file**	**Load *file* into a buffer**
Ctrl-x Ctrl-r *file*	find-file-read-only	Load *file* into a buffer for read only
Ctrl-x 2	**split-window-vertically**	**Split the current window into two windows arranged vertically**
Ctrl-x o	**other-window**	**Move the point to the other window**
Ctrl-x 3	split-window-horizontally	Split the current window into two windows arranged horizontally
Ctrl-x 0	delete-window	“Undisplay” the current window
Ctrl-x 1	**delete-other-windows**	**“Undisplay” all windows except the current one**
Ctrl-x Ctrl-b	list-buffers	Display a new window listing all buffers
Ctrl-x b *file*	switch-to-buffer	Load *file* into a buffer if necessary, and then display that buffer in the current window

Building and Debugging

Binding	Function	Description
	compile *command*	**Build the program using *command***
	gdb *executablefile*	**Launch the GDB debugger to debug *executablefile***

Miscellaneous

Binding	Function	
Ctrl-x u	undo	Undo the previous change
Ctrl-_	**undo**	**Undo the previous change**
Ctrl-g	**keyboard-quit**	**Abort the multi-keystroke command**
Ctrl-h	help-command	Access the Emacs help system
Esc `	tmm-menubar	Access the Emacs menu
Ctrl-x n	**linum**	**Display/undisplay a line number before each line**

KAIST

EE209: Programming Structures for EE

“Hello World” Program in Java, C & Python

In Java:

```
//-----------------------------------------------------------------------------------------------------------
// Hello.java
//-----------------------------------------------------------------------------------------------------------
public class Hello
{
        public static void main(String[] args)
        // Write “hello, world” to stdout.
        {
                System.out.println(“hello, world”);
        }
}
```

In C:

```
/*----------------------------------------------------------------------------------------------------------*/
/* hello.c */
/*----------------------------------------------------------------------------------------------------------*/
#include<stdio.h>
int main(int argc, char **argv)
/* Write “hello, world₩n” to stdout. Return 0. */
{
        printf(“hello, world\n”);
        return 0;
}
```

In Python:

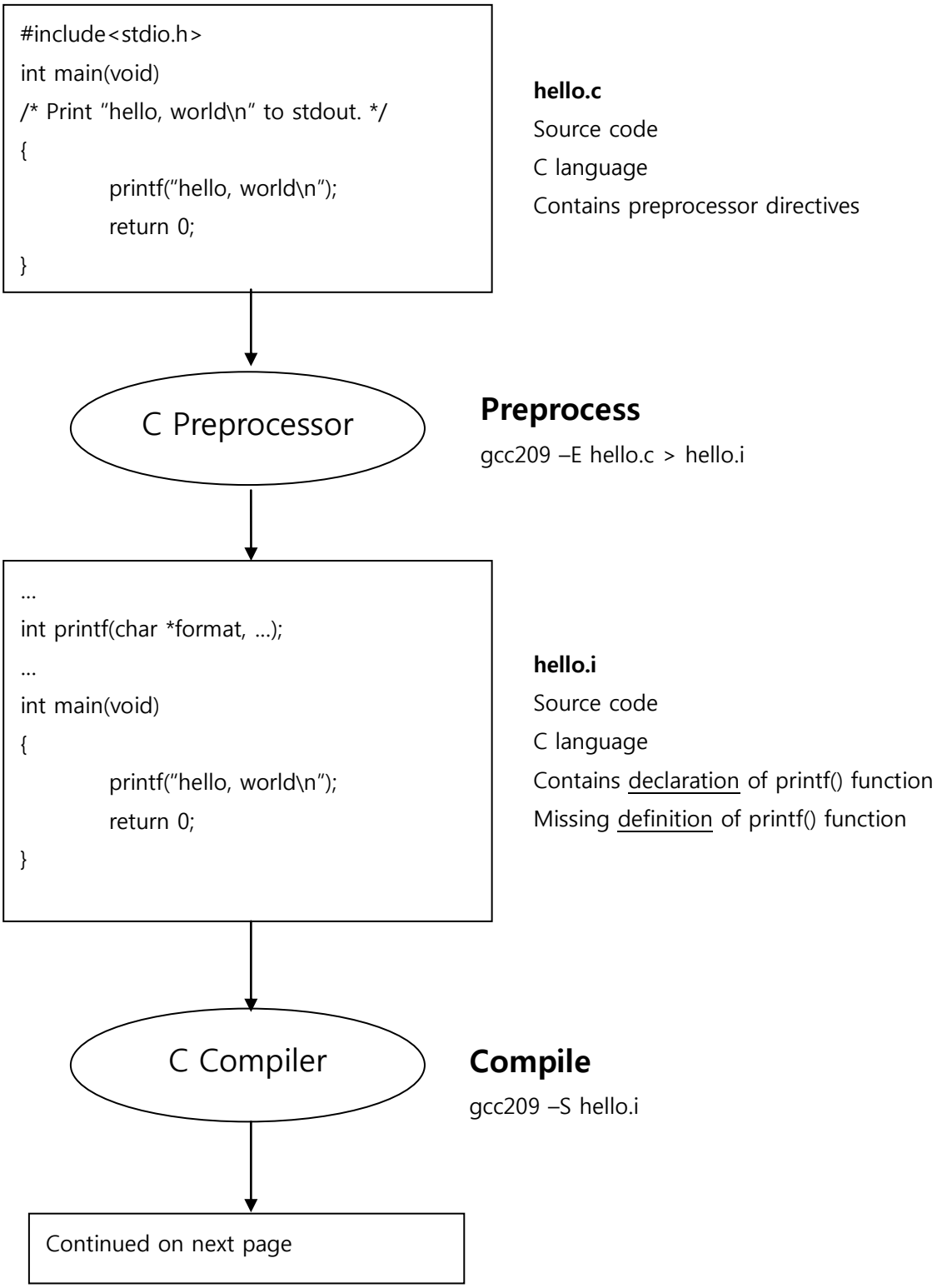
```
#----------------------------------------------------------------------------------------------------------#
# hello.py #
#----------------------------------------------------------------------------------------------------------#

print “hello, world”
```

KAIST

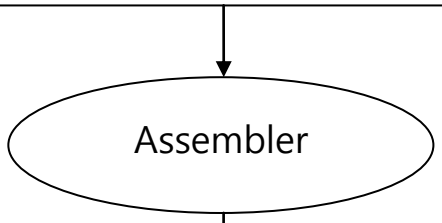
EE209: Programming Structures for EE

Building C Programs



```
    .section   .rodata
cGreeting:
    .asciz "hello, world₩n"
    .section .text
    .global main
    .type main, @function
main:
    puchl %ebp
    movl %esp, %ebp
    pushl %cGreeting
    call printf
    addl $4, $esp
    movl $0, %eax
    movl %ebp, %esp
    popl %ebp
    ret
```

hello.s
Source code
Assembly language
Missing definition of printf() function



Assemble

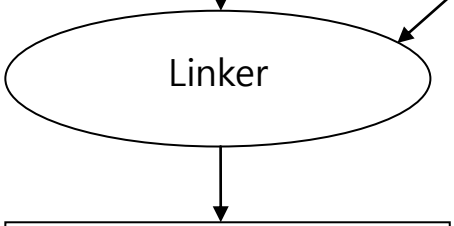
gcc209 –c hello.s

hello.o
Object code
Machine language
Missing definition of printf() function

.10010100011010010010...

11110010000010100100110...

libc.a
Library containing machine language definition of printf() function(and many others)



Link

gcc209 hello.o –lc -o hello

hello

Exchange code
Machine language

00101000010100000011111O...

Shortcut:
gcc209 hello.c –o hello

gcc209
is an abbreviation for
gcc –Wall –ansi –pedantic –m32 –march=i386