Computer Science 102
Lab 2

Objectives

- Complete the list.c module containing some simple list managment functions.
- Learn how to construct and use a simple makefile
- Learn how to use the Unix tar utility program

In this lab you will complete the implementation of the functions that comprise the *list.c* function whose prototypes are found in the file *list.h.* You must use the structure definitions also found in list.h and you should use listmain.c to test your program.

A makefile for a multi-module program:

The Unix *make* program is a handy utility that can be used to build things ranging from programs to documents. Elements of significance include:

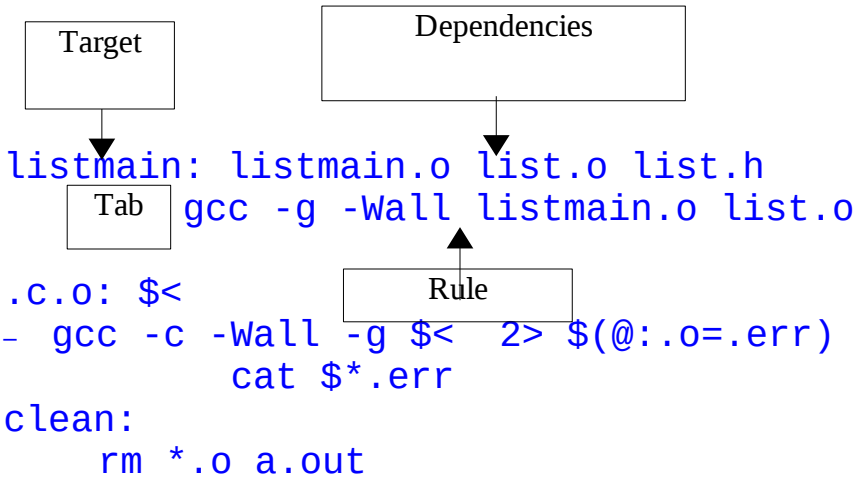
targets labels that appear in column 1 and are followed by a the character *":"* . The *make* command can take a target as an operand as in *make listmain*

dependencies are files that are enumerated following the name of the target. If any dependency is newer than the target, the target will be rebuilt.

rules are specified in lines following the target and specify the procedure for building the target. Rules *must* start with a *tab character*. In the example below the tab has been expanded as spaces *but you may not enter spaces.*

An example makefile

To use make you must first build a file named *makefile.* The following *makefile* can be used build the executable list test program. *Don't confuse a makefile with a shell-script file which is simply a text file that contains a collection of commands.*



The target .c.o: is called a *suffix rule*. It is telling *make* to use the commands that follow the suffix rule whenever it needs to convert a .c file into the corresponding .o file.

Notice that two rules (gcc and cat) follow the dependency The – that precedes the second *gcc* command allows make to continue and do the *cat* even if gcc fails. Normally make will terminate if a rule fails.

There are a number of predefined macro based names:

Macros are names that begin with the character $. When *make* processes a makefile the macro name is replaced by the current value of the macro in a manner similar to what is done in Unix script or Windows .cmd files.

$@ -- the current target's full name
$? -- a list of the target's changed dependencies
$< -- similar to $? but identifies a single file dependency and is used only in suffix rules
$* -- the target file's name without a suffix

Another handy macro based facility permits one to change suffixes on the fly. The macro $(@:.o=.err) says use the target name but change the .o to .err. Thus when you build the program you will see that files named *listmain.err* and *list.err.* These files will contain the compiler warnings and errors messages.

Building a program with *make*

Assuming you have correctly constructed the file named *makefile* in your working directory and you wish to build the first target (here the listmain executable) you simply enter the command:

```
make
```

The Unix *tar* utility

The name *tar* originally stood for *tape-archiver* because the program was originally designed to backup entire Unix directory trees to tape. Now it is commonly used to collect multiple files or entire directory trees into a single file for easy transport as an e-mail attachment. In this way it works somewhat like the Windows “zip” program but it doesn't compress anything. Linux tar can be used to compress files as well but solaris tar can't.

To create a tar file in the directory above the directory in which you are working enter the command

```
tar cvf ../list.tar .
```

to see what is in the tar file do

```
tar tvf ../list.tar
```

to extract the files from the tar file do

```
tar xvf ../list.tar
```

use care here because tar will *overwrite any existing files* with the same names as files in the archive with *no warning.* For example, the command *tar cvf* * will *wipe out the first file in alphabetic order and replace it with the tar file!!*!

The "z" flag tells linux tar to compress the tar file. To create a compressed tar file use the command:

```
tar czvf ../list.tar .
```

Submitting

In this lab you will submit a tar file named list.tar containing listmain.c list.c list.h and your makefile

```
sendlab.102.labsection#  lab#  list.tar
```

Since this is lab2 and if you are in section 3 the command you should use is (remember to cd .. because that is where you put your tarfile!)

```
sendlab.102.3 2 list.tar
```