The University of Queensland
School of Information Technology and Electrical Engineering
Semester One, 2012
CSSE2310 / CSSE7231 - Assignment 3
Due: 11:10pm 11 May, 2012
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 1.5

Introduction

Your task is to write a (c99) program (called `thresher`) which executes a specified compiler and produces a report on the number and type of errors encountered. This will require creating processes and interprocess communication. Your assignment submission must comply with the C style guide available on the course website.

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

In this course we will use the subversion (svn) system to deal with assignment submissions. Do not commit any code to your repository unless it is your own work or it was given to you by teaching staff. **If you have questions about this, please ask.**

Invocation

When run with less than three arguments (not counting `thresher` it should print usage instructions to stderr:

```
Usage: thresher [--show] type command filename ...
```

The *type* indicates both the arguments used for the build and how to identify errors. The *cmd* is the program to execute to perform the build. The remaining arguments are the names of files to be processed. The optional `--show` argument indicates that output generated by the build program should be output to `stdout`.

For example:

- `./thresher ansiC gcc bob.c`
  Would use `gcc` to compile `bob.c` using `ansiC` options and looking for C errors.
- `./thresher java javac c1.java c2.java`
  Would compile `c1.java` and `c2.java` (individually) using javac.

The build options are as shown in Table 1. Descriptions of how to identify errors are given later in this specification.

Type	**Build options**
ansiC	*prog* `-ansi -pedantic -Wall` *file*
c99	*prog* `-std=gnu99 -pedantic -Wall` *file*
java	*prog -d . file*
latex	*prog file*

Table 1: *prog* and *file* should be replaced with the relevant parameters.

Compilation

Your code must compile with command:
`make`

Each individual file must compile with at least `-Wall -pedantic -std=gnu99`. You may of course use additional flags but you must not use them to try to disable or hide warnings. You must also not use pragmas to achieve the same goal.

If any errors result from the make command (ie the executable cannot be created), then you will receive 0 marks for functionality. Any code without academic merit will be removed from your program before compilation is attempted (and if compilation fails, you will receive 0 marks for functionality). If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs apart from those passed on the command line. Your solution must not use non-standard headers/libraries.

Exit status

Condition	**Status**	**Message to stderr**
Too few parameters	1	Usage: thresher [--show] type command filename ...
Type parameter is not a one of the permitted values	2	Unknown build type
Exec of build program fails	3	Exec failed
Other system call error	4	System error
Build program exits with status !=0	5	

All other circumstances `thresher` should exit with status 0.

Reports

If the `--show` argument is given, then all output on the tool's standard error (or `stdout` in the case of LaTeX builds) should be echoed to `thresher`'s standard out.

Once each file has been processed a report should be printed to stdout. The first line should consist of 4 dashes. Any output required by the `--show` option comes next followed by a line of four dashes. Do not print this second line if not using `--show`. Then a count of each type of error that occurred. If an error type did not occur, do not print it. Next print the filename followed by either "exited with status X"[1] or "did not exit normally" if the program terminated for some other reason. Finally, print another row of 4 dashes.

No other output should be produced when your program is run.

[1]subsitute the real exit status

For example, if `gcc -ansi -pedantic -Wall err.c` outputs:

```
err.c: In function `main':
err.c:3:4: error: `for' loop initial declarations are only allowed in C99 mode
err.c:3:4: note: use option -std=c99 or -std=gnu99 to compile your code
err.c:5:7: warning: implicit declaration of function `getType' [-Wimplicit-function-declaration]
err.c:5:23: error: `sdfsdf' undeclared (first use in this function)
err.c:5:23: note: each undeclared identifier is reported only once for each function it appears in
err.c:5:31: error: `sdf' undeclared (first use in this function)
err.c:8:7: error: label at end of compound statement
err.c:11:4: error: expected expression before '/' token
err.c:13:1: warning: control reaches end of non-void function [-Wreturn-type]
```

`./thresher ansiC gcc err.c` would output:

```
----
1 implicit declaration
2 undeclared
1 c99
1 c++ comment?
2 other
err.c exited with status 1
----
```

Error Types

The following sections describe how to identify errors in the different build types. In each case, the conditions should be checked in the order given in below. Once one condition is met, do not check for any further possibilities further down.

ansiC

1. Ignore any lines which do not begin with *name*:*number*:
2. Ignore any lines which contain `note:`
3. Ignore any lines which contain `error:  (Each undeclared`
4. Ignore any lines which contain `error:  for each function`
5. A line containing: “implicit declaration” is type “implicit declaration”.
6. A line containing: “undeclared” is type “undeclared”.
7. A line containing: “C99 mode” is type “c99”
8. A line containing: “expected expression before `'/'` token” is type “c++ comment?”
9. All other lines are type “other”

C99

Follows the same sequence as ansiC but rules 7 and 8 do not apply.

java

1. Ignore any lines which do not begin with *name*:*number*:
2. Any lines which contain "<identifier> expected" are type "missing identifier"
3. Any lines which contain "cannot find symbol" are "missing symbol"
4. Any lines which contain "static context" are "non-static access"
5. Any other lines are type "other".

latex

1. Any line that contains "! Missing $ inserted." is of type "math mode error".
2. Any line that contains "! Undefined control sequence." is of type "bad macro".
3. Any containing "LaTeX Warning" is of type "warning"
4. Any containing "LaTeX Error" is of type "error"
5. Any containing "Overfull \hbox" is of type "bad box"
6. Ignore all other lines.

LaTeX expects a response to certain messages. When your program detects Item 1 or 2 it should send a newline character to the program. When it detects Item 4 it should send 'X' followed by a newline.

Regardless of the "compiler" chosen, do not send text to it unless it is actually required. That is, do not try to make life easier for yourself by sending a long string of X's to the compiler.

Style

You must follow *version 1.6* of the C programming style guide found at: `http://courses.itee.uq.edu.au/csse2310/2012s1/resources/c_resources.html` All tab characters will be replaced using the `expand` tool before assignment are marked.

Submission

Submission must be made electronically by committing using subversion. In order to mark your assignment the markers will check out `/ass3/trunk` from your repository on `svn.eait.uq.edu.au`. Code checked in to any other part of your repository will not be marked.

The due date for this assignment is given on the front page of this specification. Note that no submissions can be made more than 120 hours past the deadline under any circumstances.

Test scripts will be provided to test the code on the trunk. Students are *strongly advised* to make use of this facility after committing.

Note: Any .h or .c files in your trunk will be marked for style *even if they are not linked by the makefile.* If you need help moving/removing files in svn, then ask.

You must submit a `Makefile` or we will not be able to compile your assignment. Remember that your assignment will be marked electronically and strict adherance to the specification is critical.

Additional requirements

When `thresher` exits, it should not leave any child processes running. If `thresher` receives `SIGINT` it should terminate any child process which is running and then exit with status 0.

Marks

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, as outlined below. Partial marks will be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. If your program does not allow a feature to be tested then you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your programs should not crash or lock up/loop indefinitely.

- Correctly invalid arguments (2 marks)
- ansiC build type
  - Correct reporting for a build producing only "other" errors (8 marks)
  - Correct reporting for all error types (6 marks)
  - Implement `--show` option (4 marks)
- C99 build type (5 marks)
- java build type (5 marks)
- latex build type (10 marks)
- Correctly terminating child process on SIGINT (2 marks)

Style (8 marks)

If g is the number of style guide violations and w is the number of compilation warnings, your style mark will be the minimum of your functionality mark and:

$$8 \times 0.9^{g+w}$$

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of version 1.6 of the C Programming Style Guide. A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final — it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` and `expand(1)` tools. Your style mark can never be more than your functionality mark — this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

Late Penalties

Late penalties will apply as outlined in the course profile.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Test Data

Test data and scripts for this assignment will be made available. The idea is to help clarify some areas of the specification and to provide a basic sanity check of code which you have committed. *They are not guaranteed to check all possible problems nor are they guaranteed to resemble the tests which will be used to mark your assignments.* Testing that your assignment complies with this specification is still *your* responsibility.

Notes and tips

- Start early.
- Write simple programs to try out fork(), exec() and pipe().
- If the builder fails to execute, the output (with `--show`) should be:

  ```
  ----
  ----
  Exec failed
  ----
  ```

- Be sure you test on moss. Different versions of gcc may produce slightly different error messages.
- Your program must not call the following functions: `system()`, `popen()`, `prctl()`.
- You should not assume that system calls will always succeed.

Updates

1. Added prctl to the list of banned functions.
2. Modified the flags to javac.
3. Modified some ignore lines for ansiC to accomodate different gcc versions.
4. Fixed missing space in java error descriptions.
5. Do not attempt to build multiple files concurrently. Build the first one, print the report, then move on.
6. If any build step fails, then `thresher` should exit. Only attempt to build the next file if the previous one returned 0.

7. Tabs are now permitted in source files **BUT**, all assignments will be run through `expand` before marking.
8. Ammended Latex error conditions.
9. - (1.4) Corrected which output stream to read (stderr for everything except LaTeX)
10. Fixed more en-dash sloppiness
11. - (1.5) Fixed err.c example
12. All LaTeX error types are now any line containg. They don't need to be the only thing on the line.