COMP10002 Foundations of Algorithms

Semester Two, 2017

C Programming Tools

Overview

Separate compilation

The C preprocessor

Debugging support

Profiling

IDE's

Separate compilation

Any non-trivial C program gets written as a sequence of modules.

Each module provides a public interface, and has a private implementation that is not necessarily revealed to the calling environment.

The interface is described by a `.h` file that lists the public (exported) functions associated with the module, and the data types used to access that functionality, typically as pointers to structures.

Separate compilation

Modules can be separately compiled, without needing to have a `main` function:

```
gcc -ansi -pedantic -Wall -c -o treeops.o treeops.c
gcc -ansi -pedantic -Wall -c -o getword.o getword.c
gcc -ansi -pedantic -Wall -c -o treeeg.o treeeg.c
```

Compiled object files can then be linked together by `gcc`:

```
gcc -o treeeg treeeg.o getword.o treeops.o
```

to make an executable program.

Separate compilation

Why? Because a big program might contain tens or hundreds of thousands of lines of code, and take many minutes (or longer) to compile.

And modules only need recompilation if they are affected by whatever change has just been made to the program.

A `.o` file is dependent on the corresponding `.c` files, and on any `.h` files that it references via `#include` directives.

The final executable depends on all the `.o` files.

Separate compilation

If all the dependencies in a program are known, then a minimal set of required `gcc` commands can be issued following any set of edits to files making up the program.

When the Unix program `make` is invoked it reads a `makefile` that describes dependencies. Then it examines all of the "last modified" dates; and issues the required commands to put all dependencies into correct timestamp order.

Make and makefiles

To describe the situation in the treeeg example:

```
COPTS = -Wall -pedantic -ansi
CC = gcc
OBJS = treeeg.o treeops.o getword.o

treeeg: $(OBJS)
        $(CC) -o treeeg $(OBJS)

treeeg.o:       treeops.h getword.h

getword.o:      getword.h

treeops.o:      treeops.h

.c.o:
        $(CC) $(COPTS) -c -o $@ $<
```

Then, to compile at any stage, just type make!

Make and makefiles

Can add any rules you like:

```
tests:  test1 test2

test1:  ass2e data0.txt numb0.txt
        ass2e < data0.txt | diff - numb0.txt

test2:  ass2e ass2d
        ass2e < data1.txt | ass2d | cmp - data1.txt
        ass2e < data2.txt | ass2d | cmp - data2.txt
        ass2e < data3.txt | ass2d | cmp - data3.txt

ass2e:  ass2e.c
        gcc -Wall -ansi -pedantic -o ass2e ass2e.c
```

Running `make` will compile `ass2e` if required, and then run two tests. First target is default, but can also `make test2`.

The C preprocessor

Symbolic substitution.

So must parenthesize each instance of each argument to avoid accidental precedence issues.

Can get both literal and “string” substitutions.

Watch out for side effects.

And don’t try to be too clever.

The C preprocessor

- `preproc.c`

Debugging – gdb

The program gdb manages an executing program and provides debugging information.

Compile using the additional -g flag.

Executing gdb prog accesses additional symbol table information that connects the source code and the executable.

Debugging – gdb

Commands:

- `b` *nn*: set a breakpoint at line *nn*
- `run`: execute the program through until a breakpoint
- `continue`: continue the program until a breakpoint
- `print` *var*: print the value of *var*
- `next`: execute a single line (bypasses functions)
- `step`: execute a single step (goes into functions)
- *return*: do the last command again

Profiling – gprof

The program `gprof` reads a file `gmon.out` that is created by running a program compiled with `-pg`.

Provides a list of functions in the program and the number of times each was called, and the time spent in each.

Use it to find the hot spots that will benefit from careful tuning. No point agonizing over functions that are not dominating the execution cost.

Profiling – gprof

```
wice: touch *.c
wice: make
gcc -pg -Wall -pedantic -ansi -c -o treeeg.o treeeg.c
gcc -pg -Wall -pedantic -ansi -c -o treeops.o treeops.c
gcc -pg -Wall -pedantic -ansi -c -o getword.o getword.c
gcc -pg -Wall -pedantic -ansi -o treeeg treeeg.o treeops.o getword.o
wice: wc pg2600.txt
  65335  565450 3223373 pg2600.txt
wice: treeeg < pg2600.txt > /dev/null
wice: ls -lg gmon.out
-rw-r--r-- 1 10 1702 Oct 14 22:02 gmon.out
wice: gprof treeeg
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 59.29      0.13     0.13   575339   226.71   293.16  recursive_search_tree
 18.24      0.17     0.04  7192079     5.58     5.58  compare_string_parts
 13.68      0.20     0.03   575340    52.32    52.32  getword
  4.56      0.21     0.01   575339    17.44   310.60  search_tree
  4.56      0.22     0.01    19306   519.71   618.32  recursive_insert
  0.00      0.22     0.00    19306     0.00   618.32  insert_in_order
  0.00      0.22     0.00    19306     0.00     0.00  print_then_free
  0.00      0.22     0.00        1     0.00     0.00  free_tree
  0.00      0.22     0.00        1     0.00     0.00  make_empty_tree
  0.00      0.22     0.00        1     0.00     0.00  recursive_free_tree
  0.00      0.22     0.00        1     0.00     0.00  recursive_traverse
  0.00      0.22     0.00        1     0.00     0.00  traverse_tree
```

IDEs

Several integrated development environments have been developed for multimodule program development.

These include `Xcode` for the Mac, and `Eclipse` for both Mac and PC.

They provide a wide range of functionality that make help when developing non-trivial programs, including automatic makefile generation, built-in debugging tools, conditional compilation, cross-module type checking, pretty-printing, and language-directed editing.