GNU Make

Sharath Gopal

'C/C++' Compilation Process

- Preprocessor
  - Expand header includes, macros, etc
  - -E option in gcc to show the resulting code
- Compiler
  - Generates machine code for certain architecture
- Linker
  - Links all modules together
  - Address resolution
- Loader
  - Loads the executable to memory to start execution

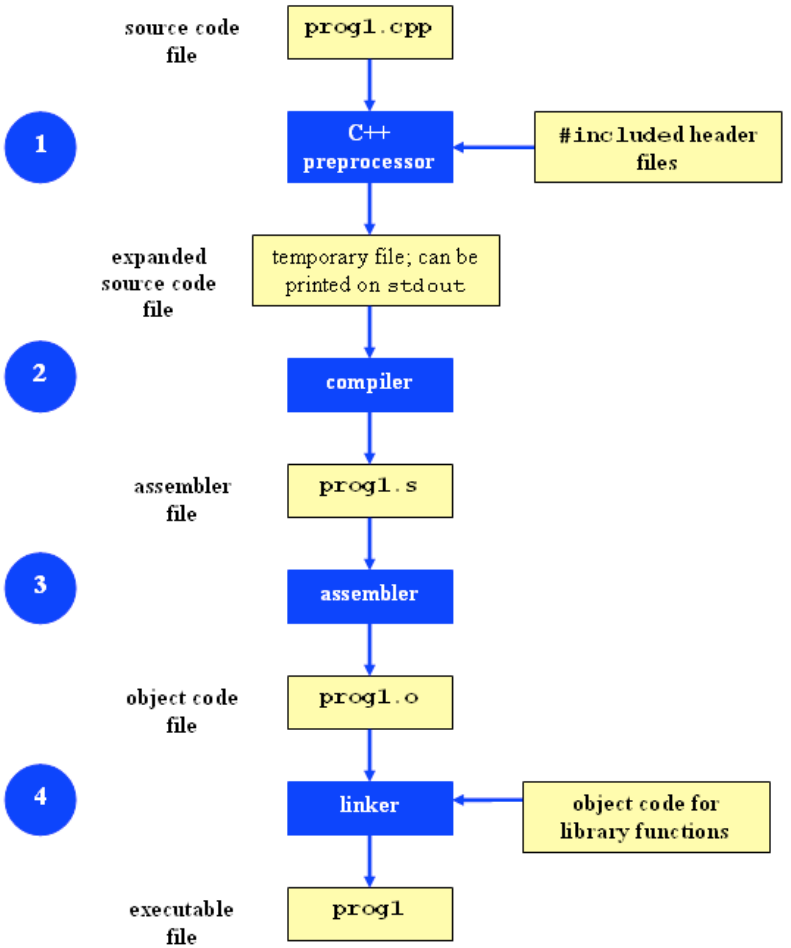

Image source - http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

GNU Make Utility

- GNU make utility “determines *automatically* which pieces of a large program need to be *recompiled*, and issues the commands to recompile them.”
  - GNU make tutorial at *http://www.gnu.org/software/make/manual/make.html*
- Makefile contains “**rules**”
  - *target …* : *prerequisites …*

    Recipe *(NOTE: You must have a tab at the beginning of the recipe)*

    …
- Once you have written the Makefile, you just have to type '**make**' at the shell prompt to start the build process from the “default rule” (the rule at the top)
- Recompiles if the object file is not present OR it is older than the source file or the dependent files.

Sample Project

foo.h

```
#ifndef __FOO_H_
#define __FOO_H_
class Foo
{
private:
  int foovar;
public:
  Foo(int _f);
  void PrintVar();
};
#endif
```

foo.cpp

```
#include "foo.h"
#include<iostream>
Foo::Foo(int _f) : foovar(_f)
{}
void Foo::PrintVar()
{
  std::cout << "Var=" << foovar << "\n";
}
```

main.cpp

```
#include<iostream>
#include "foo.h"
int main(int argc, char **argv)
{
  std::cout << "Hello World2\n";
  Foo f(40);
  f.PrintVar();
  return 0;
}
```

Sample Makefile

```
CXX=g++
mainexec : foo.o main.o
	$(CXX) foo.o main.o -o mainexec

foo.o : foo.h foo.cpp
	$(CXX) -c foo.cpp -o foo.o

main.o : foo.h main.cpp
	$(CXX) -c main.cpp -o main.o

clean :
	rm -rf *.o mainexec
```

Intrepreted Languages

- Examples: Shell, Python, Lisp, Ruby, etc
- Very good for developing quick prototypes!
- No separate compilation phase to convert high level code to machine instructions.
- Interpreter executes the program line by line and translates them to subroutines/commands already compiled into machine code.
- Stop execution if there is an error at a line
- Platform independence
- Code Indentation is important (eg:Python)
- Dynamic type checking (checked at runtime)
- Slower runtime performance compared to code that has been compiled directly to machine instructions.

Python Resources

- Python 2 docs (Chapters 1-9 should be good)
  - https://docs.python.org/2/tutorial/index.html
- Google Education – Python Course
  - https://developers.google.com/edu/python/introduction
- Comparing two strings according to LC_COLLATE
  - https://docs.python.org/2/library/locale.html
  - `locale.strcoll(string1, string2)`
- Option Parser documentation
  - https://docs.python.org/2/library/optparse.html

Python Homework

- Modify the given script randline.py to implement the comm command
  - `./comm.py -12 file1 file2`
  - `./comm.py -13 file1 -` (second file comes from STDIN)
  - `./comm.py -2 - file2` (first file comes from STDIN)
- Files sorted in the current locale (LC_COLLATE)
  - Can use `locale.strcoll(string1, string2)` to compare strings according to LC_COLLATE
- Look at the man page of `comm` to see what the options mean
  - `man comm`
- `-u` option for unsorted inputs