CS35L – Spring 2019

Slide set:	3.1
Slide topics:	Modifying and Rewriting Software, Python
Assignment:	3

How to Install Software

- Windows
  - Installshield
  - Microsoft/Windows Installer
- OS X
  - Drag and drop from .dmg mount -> Applications folder
- Linux
  - rpm(Redhat Package Management)
    - RedHat Linux (.rpm)
  - apt-get(Advanced Package Tool)
    - Debian Linux, Ubuntu Linux (.deb)
  - Good old build process
    - configure, make, make install

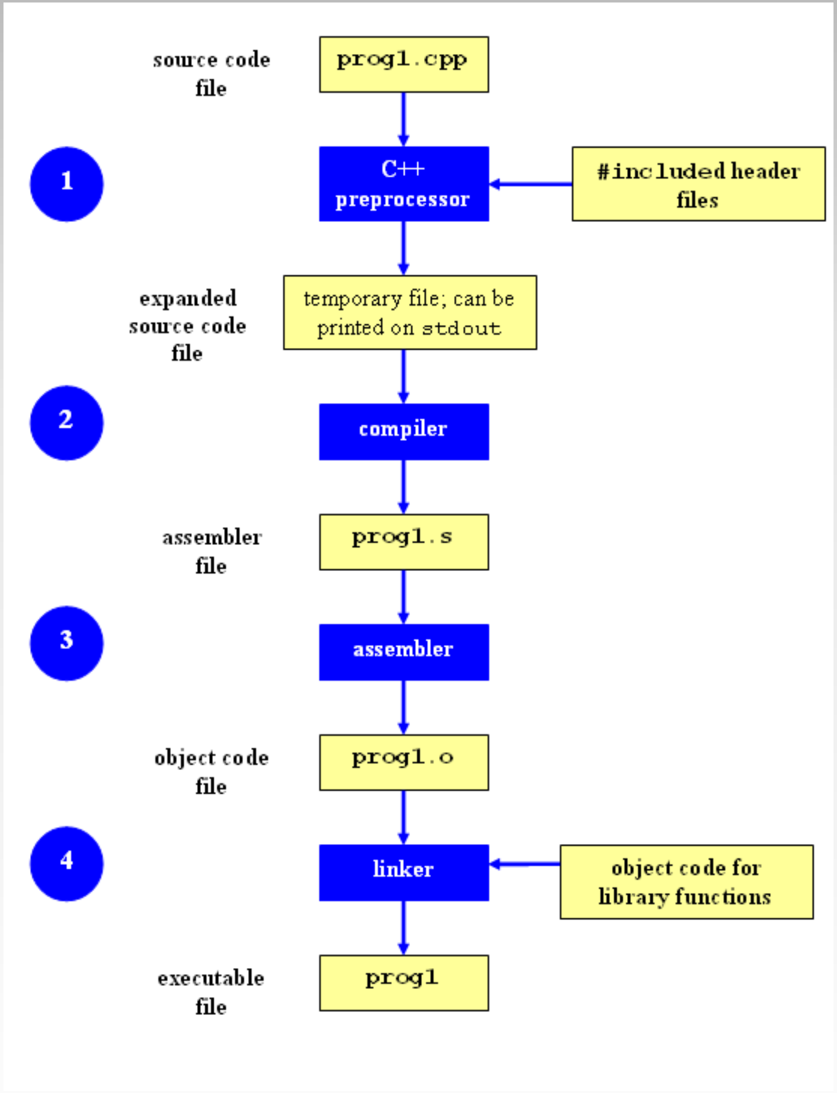
Decompressing Files

- Generally, you receive Linux software in the tarball format (.tgz) or (.gz)

Decompress file in current directory:

- $ tar –xzvf filename.tar.gz
  - Option –x: --extract
  - Option –z: --gzip
  - Option –v: --verbose
  - Option –f: --file

Compilation Process



Command-Line Compilation

- shop.cpp
  - #includes shoppingList.h and item.h
- shoppingList.cpp
  - #includes shoppingList.h
- item.cpp
  - #includes item.h
- How to compile?
  - **g++ -Wall shoppingList.cpp item.cpp shop.cpp –o shop**

What if…

- **We change one of the header or source files?**
  - Rerun command to generate new executable
- **We only made a small change to item.cpp?**
  - not efficient to recompile shoppinglist.cpp and shop.cpp
  - Solution: avoid waste by producing a separate object code file for each source file
    - g++ -Wall –c item.cpp… (for each source file)
    - g++ item.o shoppingList.o shop.o –o shop (combine)
    - Less work for compiler, saves time but more commands

What if…

- **We change item.h?**
  - Need to recompile every source file that includes it & every source file that includes a header that includes it. Here: item.cpp and shop.cpp
  - Difficult to keep track of files when project is large
    - Windows 7 ~40 million lines of code
    - Google ~2 billion lines of code

=> Make

Make

- Utility for managing large software projects
- Compiles files and keeps them up-to-date
- Efficient Compilation (only files that need to be recompiled)

Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
	g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.h
	g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.h
	g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.h shoppingList.h
	g++ -g -Wall -c shop.cpp
clean :
	rm -f item.o shoppingList.o shop.o shop
```

Rule

🟩	Comments	
🟦	Targets	Dependency Line
🟪	Prerequisites	Dependency Line
🟫	Commands	

Build Process

- **configure**
  - Script that checks details about the machine before installation
    - Dependency between packages
  - Creates ‘Makefile’
- **make**
  - Requires ‘Makefile’ to run
  - Compiles all the program code and creates executables in current temporary directory
- **make install**
  - make utility searches for a label named install within the Makefile, and executes only that section of it
  - executables are copied into the final directories (system directories)

What is Python?

- Not just a scripting language
- Object-Oriented language
  - Classes
  - Member functions
- Compiled and interpreted
  - Python code is compiled to bytecode
  - Bytecode interpreted by Python interpreter
- Not as fast as C but easy to learn, read and use
- Very popular at Google and other big companies

Why is it popular?

- Uses English keywords frequently where other use different punctuation symbols
- Fewer Syntactical Constructions
- Automatic Garbage Collection
- Easy integration with other programming languages

Different Modes

- Interactive:
  - Run commands on the python shell without actually writing a script/program.

- Script Mode:
  - Type a set of commands into a script
  - Execute all the commands at once by running the script

Python Variables

Case sensitive

Start with _ (underscore) or letters followed by other letters, underscores or digits

Other special characters are not allowed as part of the variable name

Certain reserved words may not be used as variable names on their own unless concatenated with other words

Example: Python Variables

Python Script:

```
#!/usr/bin/python
counter = 100    # An integer assignment
miles = 1000.0   # A floating point
name = "John"    # A string
print(counter)
print(miles)
print(name)
```

Output:

```
100
1000.0
John
```

Python Lines and Indentation

No braces to indicate blocks of code for class and function definitions or flow control

Blocks of code are denoted by line indentation, which is why it is strictly enforced

Number of spaces for indentation may be variable but all the statements within the same block must be equally indented

Hence, a single space has the ability to change the meaning of the code

Python Decision Making

```
#!/usr/bin/python
var = 100
if ( var == 100 ) :
        print(”Correct”)
print("Good bye!“)
```

Python List

- Common data structure in Python
- A python list is like a C array but much more:
  - **Dynamic (mutable)**: expands as new items are added
  - **Heterogeneous:** can hold objects of different types
- How to access elements?
  - List_name[index]

Example

- >>> t = [123, 3.0, ‘hello!’]
- >>> print t[0]
  - 123
- >>> print t[1]
  - 3.0
- >>> print t[2]
  - hello!

Example – Merging Lists

- >>> list1 = [1, 2, 3, 4]
- >>> list2 = [5, 6, 7, 8]
- >>> merged_list = list1 + list2
- >>> print(merged_list)
  - **Output: [1, 2, 3, 4, 5, 6, 7, 8]**

Python Dictionary

- Essentially a hash table
  - Provides key-value (pair) storage capability
- Instantiation:
  - d = {}
  - This creates an EMPTY dictionary
- Keys are unique, values are not!
  - Keys must be immutable (strings, numbers, tuples)

Example

```
d = {}
d[‘france’] = “paris”
d[‘japan’] = “tokyo”
print(d[‘france’])

d[‘germany’] = “berlin”
if (d[‘france’] == ”paris”):
     print(“Correct!”)
else:
     print(“Wrong!”)
del d[‘france’]
del d
```

for loops

```
list1 = ['Mary', 'had', 'a', 'little', 'lamb']
```

```
for i in list1:
        print i
```

Result:

```
Mary
had
a
little
lamb
```

```
for i in range(len(list1)):
        print i
```

Result:

```
0
1
2
3
4
```

I/O Basics

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline)

- *s = raw_input("Enter your input: ");*
- *print("Received input is : ", s)*

The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

- *str = input("Enter your input: ");*
- *print(“Received input is : ", str)*