CS 246 - Object Oriented Programming **Fall 2016**

Lecture 2: September 15, 2016

Lecturer: Brad Lushman *Notes By: Harsh Mistry*

2.1 Shell Scripts

Shell scripts are files containing sequences of commands, executed as programs.

Example 2.1 -

```
#!/bin/bash|
date
whoami
pwd
```

- `#!/bin/bash` : is the shebang line and tells the shell to execute this as a bash script

Note

In order to execute the shell scripts, you must have execute permissions on the file. Execute the following the assign execute permissions on the file : chmod utx myscript

In addition, to ensure bash can execute the application, a `./` must be placed before the program name

2.1.1 Variables

- Variable Declaration : `x=1` (No Spaces)
- Fetching Variable : `$x` or `${x}`
- All variables contain strings

2.1.1.1 Global Variables

By using the command `env`, a list of global variables can be viewed.

Most importantly, the `PATH` global variable is critical, as it stores a list of directories searched by bash inorder to determine commands that it can execute.

Special vars for scripts

Special Vars allow for input to be received from parameters included on the original command call. Special vars are : `$1, $2, $3, $3, etc`

2.1.2 If Statements

Example 2.2 *Check whether a word is in a dictionary*

```
#!/bin/bash
egrep "^$1$" /usr/shar/dict/words
```

Example 2.3 *A good password should not be in the dictionary. Answer if a word is a good password.*

```
#!/bin/bash
egrep "^$1$" /usr/shar/dict/words > /dev/null
if [ $? -eq 0 ]; then
    echo Bad Password
else
    echo Maybe  a Good Password
fi
```

Example 2.4 *Verify # of args, print error msg, if wrong number of arguments is provided*

```
#!/bin/bash
usage() {
    echo "Usage: $0 password " >&2
}
if [ $# -ne 1 ]; then
    usage
    exit 1
fi
```

Note

Status Codes :
Every program returns a status code when finished.

- egrep returns 0 if found, 1 if not found.

(In linux : 0 = success, non-zero = failure)

Notable Commands and Variables :

- `/dev/null` : Suppresses Output
- `$?` : Status of most recently executed command
- `if[... ... ...]; then` : Runs a program and returns 0 if program passed based on input parameters

2.1.3 If Statement General Form

```
if [ cond ]; then
    ...
elif [ cond ]; then
    ...
else
    ...
fi
```

2.1.4 Loops

Example 2.5 *Print # 's from 1 to $ 1*

```
x=1
while [ x -le $1 ]; do
    echo $x
    x=$((x+1))
done 00
```

Example 2.6 *Rename all .cpp files to .cc*

```
#!/bin/bash
for name in *.cpp; do
    mv ${name} ${name%cpp}cc
done
```

Example 2.7 *How many times does word $1 occur in file $2?*

```
#!/bin/bash
x=0
for word in $(cat $2); do
    if [ "$word" = "$1" ]; then
        x = $((x+1))
    fi
done
echo  $x
```

Example 2.8 *Payday is the last friday of the month. When is the first payday?*

- *Compute payday*
- *Report answer*

```
#!/bin/bash
answer(){
    if [ $1 -eq 31 ]; then
        echo "This month is the 31st"
    else
        echo "This month : the ${1}th"
    fi
}
answer $(cal | awk '{print $6}' | egrep "[0-9]" | tail -1)
```

Example 2.9 *Generate payday to any month*

```
#!/bin/bash
answer(){
    if [ $2 ]; then
        preamble=$2
    else
        preamble="This Month"
    fi

    if [ $1 -eq $1]; then
        echo "${preamble}: the 31st"
    else
        echo "${preamble}: the ${1}th"
    fi
}
answer $(cal $1 $2 | awk '{print $6}' | egrep "[0-9]" | tail -1) $1
```

Notable Commands and Practices

- `___ $((___))` : Enables Arithmetic
- `mv` : Rename
- `${name%cpp}` : Removes all trailing characters that match
- `"$1"` : Quotes around variables not set by you is good practice to ensure the variable value does not conflict with formatting
- `cal month year`: Prints a calender

2.2 Software Engineering Topic : Testing

- Essential part of program development
- ongoing and not just at the end
  - Testing begins before you start coding
  - Test suites will assess the expected behaviour
- Testing is not debugging. Testing simply just informs you of the bugs within your program.
- Testing cannot guarantee correctness, as it only proves ”wrongness”
- Ideally, developer and tester should be different people, unfortunately this is **CS246** where the profs and ISAs don't care for your well being

2.2.1 Types of testing

- Human testing
  - humans look over code and find flaws
  - Consists of code inspections and walkthroughs
- Machine testing
  - Runs the program on selected input and check against specifications
  - Can not check everything , so you must choose test cases carefully
- Black/White/Grey Box testing - no/full/Some knowledge of program implementation
  - Black Box Testing
    - Various classes of input (i.e numeric ranges, positive v.s input, etc)
    - Boundaries of valid data ranges (Edge Cases)
    - Multiple simultaneous boundaries (Corner Cases)
    - Intuition is often helpful with guessing likely errors
    - Extreme cases
  - White Box Testing

  - ∗ Execute all logical paths through the program
  - ∗ Make sure every function runs

- Performance testing - Allows for the tester to check if the program efficient enough for the desired application
- Regression testing - Ensures new changes to the program do not break old test cases

2.3 Module 2 : C++

Hello World is **Cool**, so why not do a quick comparison between the two languages .

Example 2.10 *In C :*

```
#include <stdio.h>
int main(){
  printf("Hello world!");
  return 0;
}
```

Example 2.11 *In C ++*

```
#include <instream>
using namespace std;
int main(){
  cout << "Hello world" << endl;
  return 0;
}
```

Notes

- Main must return int in C++ (If return 0 is left out, C++ returns 0)
- stdio.h and printf are still available in c++
- In C++ using the header `<iostream>` is preferred for I/O
-

2.3.1 Compiling C++ Code

Compile a program file :
`g++ -5 -std=c++14 program.cc -o programexecutablename`

Compile a program file with defined parameters in /.profile :
`G++14 program.cc -o programexecutablename`

2.3.2 Input/Output

C++ has 3 I/O streams :

- **cin** : for reading from stdin
- **cout, cerr** : for printing to stdout, stderr

In addition, C++ has 2 I/O operators :

- << : "Put to" (Output)
- >> : "Get from" (Input)

All examples after this point will omit default headers

Example 2.12 *Adding 2 numbers and outputting result*

```
int main(){
    int x,y;
    cin >> x >> y;
    cout << x + y << endl;
}
```

Notes on cin

- `cin` Ignores all white spaces.
- If input for a integer is not a int, then variable is undefined.
- If the read fails, then `cin.fail()` will be true.
- If EOF is reached, then `cin.eof(), cin.fail()` will both be true
  - Both will not return true until the attempted read fails!

Example 2.13 *Read all ints from stdin and echo them one per line to stdout. In addition the program will stop on bad input or EOF.*

```
int main(){
    int i;
    while (true){
        cin >> i;
        if (cin.fail()) break;
        cout << i << endl
    }
}
```

Note

- There is an implicit conversion from cin to bool
  - Lets cin be used as a condition

Example 2.14 *Read all ints from stdin and echo them one per line to stdout. In addition the program will stop on bad input or EOF using the implicit conversion*

```
int main(){
    int i;
    while (true){
           cin >> i;
           if (!cin) break;
           cout << i << endl
    }
}
```

Bitshifting in C and its relation to C++

- >> is C's right "bitshift" operator
  - a >> b : Shifts a's bits b spots to the right
- When left-hand operand is cin, >> is the "get from" operator
- Operator >>
  - Inputs : `cin` (istream and `data` (Variety of types)
  - Output : returns cin (istream)
  - This is why we can write `cin >> x >> y >> z;`

Example 2.15 *Read all ints from stdin and echo them one per line to stdout. In addition the program will stop on bad input or EOF using implicit conversion and output of* `cin`

```
int main(){
    int i;
    while (true){
           cin >> i;
           if (!(cin >> i)) break;
           cout << i << endl;
    }
}
```

Example 2.16 *Read all ints from stdin and echo them one per line to stdout. In addition the program will stop on bad input or EOF using implicit conversion and output of* `cin` *without while(true)*

```
int main(){
    int i;
    while (cin >> i){
           if (cin >> i) break;
           cout << i << endl;
    }
}
```

Example 2.17 *Read all ints and echo to stdout until EOF. Also Skip all non-integer input.*

```
int main(){
   while(true){
        if(!(cin >> i)){
             if(cin.eof())break;
             cin.clear(); // clears the fail bit
             cin.ignore(); // ignore and throwaway current input character
        }
   }
   else cout << i << endl; // read was ok
}
```