Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

EECS2031

Introduction

Introduction

- Instructor: Mokhtar Aboelaze
- Room 2026 CSEB
  lastname@cse.yorku.ca x40607
- Office hours W 2:00-4:00 or by appointment

Grading Details

- Participation 5%
- Lab 15%
- 3 tests 18% each (total 54%)
- Final 26%

About the course

- By the end of the course, the students will be expected to be able to:
  - Use the basic functionality of the Unix shell, such as standard commands and utilities, input/output redirection, and pipes
  - Develop and test shell scripts of significant size.
  - Develop and test programs written in the C programming language.
  - Describe the memory management model of the C programming language

Introduction

- Course Content
- C
  - Learn how to write test, and debug C programs.
- UNIX (LINUX)
  - Using Unix tools to automate making and testing.
  - Unix shell programming

Text

- The C Programming Language, Kernighan and Ritchie (K+R)
- C Programming: A Modern Approach 2nd edition K.N. King (optional)
- Practical Programming in the UNIX Environment, edited by W. Sturzlinger
- Class notes (Slides are not complete, some will be filled in during class).
- Man pages

Course Objective

- By the end of the course, you should be able to
  - Write applications (though small) in C
  - Test and debug your code
  - Use UNIX to automate the compilation process
  - Write programs using UNIX shell scripts and awk

WHY C and UNIX

- Wide use, powerful, and fast
- Both started at AT&T Bell Labs
- UNIX was written in assembly, later changed to C
- Many variants of UNIX

WHY C and UNIX

- The first part of the course is C
- The second part shell script (sh)
- We will start with a quick introduction to Unix to be able to start the labs.
- Lab 1 is this week (introduction to Unix)
- Lab policy

Introduction to Unix

- Please check the tutorial at http://www.cs.sfu.ca/~ggbaker/reference/unix/
- The first 4 tutorials
- Blackboard

C – A History

- In 1972 Kernighan and Ritchie invented C
- In 1978 Brian Kernighan and Dennis Ritchie Published their “white” book. Became defacto standard for C known as K&R C.
- ANSI completed a standard for C approved in 1989 as ANSI X3.159-1989 known as C89 or C90 (ANSI-C).
- C99 became standard in ISO/IEC 9899:1999.

Languages based on C

- C++ basically object oriented C
- Java C syntax, much more restrictive + garbage collection
- C#
- Perl started as scripting language, overtime adopted many features of C

C

- Almost low level, small, permissive (assumes you know what are you doing) language.
- Efficient, portable, powerful, and flexible (from system programming to embedded systems).
- Can be error prone, difficult to understand (see next slide)

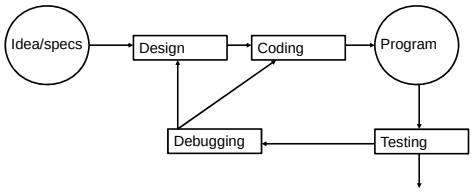
Obfuscated C

```
int v,i,j,k,l,s,a[99];
main(){
for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k
=i<s,j+=(v=j<s&&(!k&&!!printf(2+"\n\n%c"
-(!l<<!j)," #Q"[l^v?(l^j)&1:2])&&++l ||
a[i]<s&&v&&v-i+j&&v+i-j))&&!( l%=s),v||
(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i])
        ;
}
```

Tips

- Use tools to make programs more reliable
- Use existing code library
- Adopt a sensible set of coding conventions
- Avoid tricks and overly complex code (do not ever do something like the Q8.c)

Software Development Cycle



Why Testing

- **Specifications = LAW**, you have to obey it.
- No changes (*improvement)* unless it is approved
- If in doubt, ask
- First create test cases, test, if error, debug, repeat
- Testing can show the presence of faults, not their absence -- Dijkstra
- Testing is very costly, in large commercial software 1-3 bugs per 100 line of code.

Why Testing

- 1990 AT&T long distance calls fail for 9 hours
  - Wrong location for C break statement
- 1996 Ariane rocket explodes on launch
  - Overflow converting 64-bit float to 16-bit integer
- 1999 Mars Climate Orbiter crashes on Mars
  - Missing conversion of English units to metric units
- Therac: A radiation therapy machine that delivered massive amount of radiations killing at lease 5 people
  - Among many others, the reuse of software written for a machine with hardware interlock. Therac did not have hardware interlock.

Why Testing

– Jan 13, 2005, LA Times

"A new FBI computer program designed to help agents share information to ward off terrorist attacks may have to be scrapped, forcing a further delay in a four-year, half-billion-dollar overhaul of its antiquated computer system… Sources said about $100 million would be essentially lost if the FBI were to scrap the software…"

Compile and Run

- The code is compiled by using gcc
- `gcc file.c`
- `gcc –o output file.c`
- `gcc –Idirectory file.c`
- $PATH and a.out

Type of Errors

- Errors in program called bugs
- Testing is the process of looking for errors, debugging if found
- Three types of errors
  - Syntax
  - Run-time
  - Logic

Syntax Errors

- Mistakes by violating “grammar” rules
- Diagnosed by C++ compiler
- Must fix before compiler will translate code

Syntax Errors

- #<include stdio.h>
- int main ( );
- {
- printf(“Hello World”);
- /* Next line will output
- a name! /*
- printf(“ Total is %d \n”,total);
- printf(“Final result is \n,result);
- }

```
#include <stdio.h>
int main()
{
printf(“Hello World”);
/*next line will output
A name */
Printf(“Total is %d
\n”,total);
printf(“Final result is
\n”,result););
}
```

Runtime Errors

- Violation of rules during execution of program
- Computer displays message during execution and execution is terminated
- Error message may help locating error
- E.g. X= 5 / 0;

Logical Errors

- Will not be detected by the compiler, may or may not produce an error message (if it results in a runtime error)
- Difficult to find
- Execution is complete but output is incorrect
- Programmer checks for reasonable and correct output