CIS 330: Unix and C/C++

Lecture 5: finish lecture 4, memory errors
Announcements

• Friday discussion:
  – Andy will talk about ssh / scp, gdb, and valgrind
Review
Build: The Actors

• File types
  – Source code
  – Object code
  – Executable code

• Programs
  – Compiler
  – Linker
Analogy

Source Code

Object Code

Executable Code
Compilers, Object Code, and Linkers

• Compilers transform source code to object code
  – Confusing: most compilers also secretly have access to linkers and apply the linker for you.

• **Object code**: statements in machine code
  – not executable
  – intended to be part of a program

• Linker: turns object code into executable programs
Our first gcc program: named output

```
C02LN00GFD58:CIS330 hank$ cat t.c
#include <stdio.h>
int main()
{
    printf("hello world!\n");
}
C02LN00GFD58:CIS330 hank$ gcc t.c
C02LN00GFD58:CIS330 hank$ ./a.out
hello world!
C02LN00GFD58:CIS330 hank$ gcc -o helloworld t.c
C02LN00GFD58:CIS330 hank$ ./helloworld
hello world!
C02LN00GFD58:CIS330 hank$ ls -l helloworld
-rwxr-xr-x 1 hank staff 8496 Apr 3 15:15 helloworld
C02LN00GFD58:CIS330 hank$
```

"-o" sets name of output
Output name is different
Output has execute permissions
gcc flags: debug and optimization

- "gcc –g": debug symbols
  - Debug symbols place information in the object files so that debuggers (gdb) can:
    - set breakpoints
    - provide context information when there is a crash
- "gcc –O2": optimization
  - Add optimizations ... never fails
- "gcc –O3": provide more optimizations
  - Add optimizations ... sometimes fails
- "gcc –O3 –g"
  - Debugging symbols slow down execution ... and sometimes compiler won’t do it anyways...
Large code development

Why could this be a good idea?
Multi-file development: example

```
fawcett:330 childs$ cat t1.c
int doubler(int x)
{
    return 2*x;
}
fawcett:330 childs$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childs$ gcc -c t1.c
fawcett:330 childs$ gcc -c t2.c
fawcett:330 childs$ gcc -o both t2.o t1.o
fawcett:330 childs$ ./both
fawcett:330 childs$ echo $?
10
```

- `cat` is a Unix command that prints the contents of a file.
- `$?` is a shell construct that has the return value of the last executed program.
Multi-file development: example

```c
fawcett:330 childds$ cat t1.c
int doubler(int x)
{
    return 2*x;
}
fawcett:330 childds$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childds$ gcc -c t1.c
fawcett:330 childds$ gcc -c t2.c
fawcett:330 childds$ gcc -o both t2.o t1.o
fawcett:330 childds$ ./both
fawcett:330 childds$ echo $?
10
```
Linker order matters for some linkers (not Macs). Some linkers need the .o with “main” first and then extract the symbols they need as they go. Other linkers make multiple passes.
Libraries

• Library: collection of “implementations” (functions!) with a well defined interface

• Interface comes through “header” files.

• In C, header files contain functions and variables.
  – Accessed through “#include <file.h>”
Finish Lecture 4
Includes and Libraries

• gcc support for libraries
  – “-I”: path to headers for library
    • when you say “#include <file.h>, then it looks for file.h in the directories -I points at
  – “-L”: path to library location
  – “-lname”: link in library libname
Library types

• Two types:
  – static and shared

• Static: all information is taken from library and put into final binary at link time.
  – library is never needed again

• Shared: at link time, library is checked for needed information.
  – library is loaded when program runs

More about shared and static later ... for today, assume static
Making a static library

Note the ‘#’ is the comment character

C02LN00GF58:multiplier hank$ cat multiplier.h # here's the header file
int doubler(int);
int tripler(int);
C02LN00GF58:multiplier hank$ cat doubler.c # here's one of the c files
int doubler(int x) {return 2*x;}
C02LN00GF58:multiplier hank$ cat tripler.c # here's the other c files
int tripler(int x) {return 3*x;}
C02LN00GF58:multiplier hank$ gcc -c doubler.c # make an object file
C02LN00GF58:multiplier hank$ ls doubler.o # we now have a .o
  doubler.o
C02LN00GF58:multiplier hank$ gcc -c tripler.c
C02LN00GF58:multiplier hank$ ar r multiplier.a doubler.o tripler.o
C02LN00GF58:multiplier hank$ (should have called this libmultiplier.a)
What’s in the file?

```bash
C02LN00GFD58: multiplier hank$ nm multiplier.a

multiplier.a(doubler.o):
000000000000000038 s EH_frame0
000000000000000000 T _doubler
000000000000000050 S _doubler.eh

multiplier.a(tripler.o):
000000000000000030 s EH_frame0
000000000000000000 T _tripler
000000000000000048 S _tripler.eh
C02LN00GFD58: multiplier hank$
```
Typical library installations

• Convention
  – Header files are placed in “include” directory
  – Library files are placed in “lib” directory

• Many standard libraries are installed in /usr
  – /usr/include
  – /usr/lib

• Compilers automatically look in /usr/include and /usr/lib (and other places)
Installing the library

```
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier/include
C02LN00GFD58:multiplier hank$ cp multiplier.h ~/multiplier/include/
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier/lib
C02LN00GFD58:multiplier hank$ cp doubler.c multiplier.a tripler.c
doubler.o multiplier.h tripler.o
C02LN00GFD58:multiplier hank$ cp multiplier.a ~/multiplier/
C02LN00GFD58:multiplier hank$ mv multiplier.a libmultiplier.a
C02LN00GFD58:multiplier hank$ cp libmultiplier.a ~/multiplier/lib/
```

“mv”: unix command for renaming a file
Example: compiling with a library

C02LN00GFD58:CIS330 hank$ cat t.c
#include <multiplier.h>
#include <stdio.h>
int main()
{
    printf("Twice 6 is %d, triple 6 is %d\n", doubler(6), tripler(6));
}
C02LN00GFD58:CIS330 hank$ gcc -o mult_example t.c -I/Users/hank/multiplier/include -L/Users/hank/multiplier/lib -lmultipler
C02LN00GFD58:CIS330 hank$ ./mult_example
Twice 6 is 12, triple 6 is 18

• gcc support for libraries
  – "-I": path to headers for library
  – "-L": path to library location
  – "-lname": link in library libname
Makefiles

• There is a Unix command called “make”
• make takes an input file called a “Makefile”
• A Makefile allows you to specify rules
  – “if timestamp of A, B, or C is newer than D, then carry out this action” (to make a new version of D)
• make’s functionality is broader than just compiling things, but it is mostly used for compilation

Basic idea: all details for compilation are captured in a configuration file ... you just invoke “make” from a shell
• Reasons Makefiles are great:
  – Difficult to type all the compilation commands at a prompt
  – Typical develop cycle requires frequent compilation
  – When sharing code, an expert developer can encapsulate the details of the compilation, and a new developer doesn’t need to know the details … just “make”
Makefile syntax

- Makefiles are set up as a series of rules
- Rules have the format:
  target: dependencies
  [tab] system command
Makefile example: multiplier lib

C02LN00GFD58:code hank$ cat Makefile
lib: doubler.o tripler.o
  ar r libmultiplier.a doubler.o tripler.o
  cp libmultiplier.a ~/multiplier/lib
  cp multiplier.h ~/multiplier/include

doubler.o: doubler.c
  gcc -c doubler.c

tripler.o: tripler.c
  gcc -c tripler.c
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$ make
  ar r libmultiplier.a doubler.o tripler.o
  cp libmultiplier.a ~/multiplier/lib
  cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$ touch doubler.c
C02LN00GFD58:code hank$ make
  gcc -c doubler.c
  ar r libmultiplier.a doubler.o tripler.o
  cp libmultiplier.a ~/multiplier/lib
  cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
Fancy makefile example: multiplier lib

```bash
C02LN00GFD58:code hank$ cat Makefile
CC=gcc
CFLAGS=-g
INSTALL_DIR=~/.multiplier

AR=ar
AR_FLAGS=r

SOURCES=doubler.c tripler.c
OBJECTS=$(SOURCES:.c=.o)

lib: $(OBJECTS)
    $(AR) $(AR_FLAGS) libmultiplier.a $(OBJECTS)
    cp libmultiplier.a $(INSTALL_DIR)/lib
    cp multiplier.h $(INSTALL_DIR)/include

.c.o:
    $(CC) $(CFLAGS) -c $<
C02LN00GFD58:code hank$ touch doubler.c
C02LN00GFD58:code hank$ make
gcc -g -c doubler.c
ar r libmultiplier.a doubler.o tripler.o
cp libmultiplier.a ~/.multiplier/lib
cp multiplier.h ~/.multiplier/include
C02LN00GFD58:code hank$
```
Configuration management tools

• Problem:
  – Unix platforms vary
    • Where is libX installed?
    • Is OpenGL supported?

• Idea:
  – Write program that answers these questions, then adapts build system
    • Example: put “-L/path/to/libX -lX” in the link line
    • Other fixes as well
Two popular configuration management tools

• Autoconf
  – Unix-based
  – Game plan:
    • You write scripts to test availability on system
    • Generates Makefiles based on results

• Cmake
  – Unix and Windows
  – Game plan:
    • You write .cmake files that test for package locations
    • Generates Makefiles based on results

CMake has been gaining momentum in recent years, because it is one of the best solutions for cross-platform support.
Project 1C Overview
Project 1C

== Build a Makefile for math330 ==

Your Makefile should:
(1) create an include directory
(2) copy the Header file to the include directory
(3) create a lib directory
(4) compile the .c files in trig and exp as object files (.o’s)
(5) make a library
(6) install the library to the lib directory
(7) compile the “cli” program against the include and library directory

== Extend the math330 library ==

You should:
(1) add 3 new functions: arccos, arcsin, and arctan (each in their own file)
(2) Extend the “cli” program to support these functions
(3) Extend your Makefile to support the new functions
Proj 1C

• Let’s look at the prompt
• Due Mon April 17th
Tar
Unix command: tar

• Anyone know what tar stands for?

tar = tape archiver

IBM tape library
Unix command: tar

• Problem: you have many files and you want to...
  – move them to another machine
  – give a copy to a friend
  – etc.

• Tar: take many files and make one file
  – Originally so one file can be written to tape drive

• Serves same purpose as “.zip” files.
Project 4A

• Posted now
• You will practice debugging & using a debugger
  – There are 3 programs you need to debug
    • In this case, “debug” means identify the bug
      – Does not mean fix the bug
    • Can use gdb or lldb
    • May want to run on ix

• Worksheet due in class on Weds April 19th
Prep for 4A

• 4A assumes C knowledge you don’t have
• So lecture on it now
• Topics
  – Character strings
  – Memory Errors
  – Misc.
Character Strings
There have been various extensions to ASCII ...
now more than 128 characters
Many special characters are handled outside this convention
signed vs unsigned chars

• signed char ("char"):  
  – valid values: -128 to 127  
  – size: 1 byte  
  – used to represent characters with ASCII  
    • values -128 to -1 are not valid

• unsigned char:  
  – valid values: 0 to 255  
  – size: 1 byte  
  – used to represent data
character strings

• A character “string” is:
  – an array of type “char”
  – that is terminated by the NULL character

• Example:
  char str[12] = “hello world”;
  – str[11] = ‘\0’ (the compiler did this automatically)

• The C library has multiple functions for handling strings
Character strings example

```
128-223-223-72-wireless:330 hank$ cat string.c
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char *str2 = str+6;

    printf("str is \"%s\" and str2 is \"%s\"\n", str, str2);

    str[5] = '\0';

    printf("Now str is \"%s\" and str2 is \"%s\"\n", str, str2);
}

128-223-223-72-wireless:330 hank$ gcc string.c
128-223-223-72-wireless:330 hank$ ./a.out
str is "hello world" and str2 is "world"
Now str is "hello" and str2 is "world"
```
Useful C library string functions

- `strcpy`: string copy
- `strncpy`: string copy, but just first N characters
- `strlen`: length of a string

```c
#include <string.h>
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char str2[6], str3[7];
    strcpy(str2, str+strlen("hello "));
    strncpy(str3, str, strlen("hello "));
    printf("%s,%s\n", str2, str3);
}
```

```
128-223-223-72-wireless:330 hank$ cat strcpy.c
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello
```
Useful C library string functions

- `strcpy`: string copy
- `strncpy`: string copy, but just first N characters
- `strlen`: length of a string

```
128-223-223-72-wireless:330 hank$ cat strcpy.c
#include <string.h>
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char str2[7], str3[6];
    strcpy(str2, str+len("hello "));
    strncpy(str3, str, str+len("hello "));
    printf("%s,%s\n", str2, str3);
}
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello world
```
# More useful C library string functions

## Functions

### Copying:
- **memcpy**: Copy block of memory ([function](source:cplusplus.com))
- **memmove**: Move block of memory ([function](source:cplusplus.com))
- **strcpy**: Copy string ([function](source:cplusplus.com))
- **strncpy**: Copy characters from string ([function](source:cplusplus.com))

### Concatenation:
- **strcat**: Concatenate strings ([function](source:cplusplus.com))
- **strncat**: Append characters from string ([function](source:cplusplus.com))

### Comparison:
- **memcmp**: Compare two blocks of memory ([function](source:cplusplus.com))
- **strcmp**: Compare two strings ([function](source:cplusplus.com))
- **strcoll**: Compare two strings using locale ([function](source:cplusplus.com))
- **strn cmp**: Compare characters of two strings ([function](source:cplusplus.com))
- **strxfrm**: Transform string using locale ([function](source:cplusplus.com))

### Searching:
- **memchr**: Locate character in block of memory ([function](source:cplusplus.com))
- **strchr**: Locate first occurrence of character in string ([function](source:cplusplus.com))
- **strcspn**: Get span until character in string ([function](source:cplusplus.com))
- **strpbrk**: Locate characters in string ([function](source:cplusplus.com))
- **strrchr**: Locate last occurrence of character in string ([function](source:cplusplus.com))
- **strspn**: Get span of character set in string ([function](source:cplusplus.com))
- **strstr**: Locate substring ([function](source:cplusplus.com))
- **strtok**: Split string into tokens ([function](source:cplusplus.com))

### Other:
- **memset**: Fill block of memory ([function](source:cplusplus.com))
- **strerror**: Get pointer to error message string ([function](source:cplusplus.com))
- **strlen**: Get string length ([function](source:cplusplus.com))

## Macros
- **NULL**: Null pointer ([macro](source:cplusplus.com))

## Types
- **size_t**: Unsigned integral type ([type](source:cplusplus.com))

[source: cplusplus.com]
Memory Errors
Memory Errors

• Array bounds read
  ```
  int main()
  {
    int var;
    int arr[3] = { 0, 1, 2 };  
    var=arr[3];
  }
  ```

• Array bounds write
  ```
  int main()
  {
    int var = 2;
    int arr[3];
    arr[3]=var;
  }
  ```
Memory Errors

Vocabulary: “dangling pointer”: pointer that points to memory that has already been freed.

```c
int main()
{
    int *var = malloc(sizeof(int)*2);
    var[0] = 0;
    var[1] = 2;
    free(var);
    var[0] = var[1];
}
```

When does this happen in real-world scenarios?
Memory Errors

• Freeing unallocated memory

```c
int main()
{
    int *var = malloc(sizeof(int)*2);
    var[0] = 0;
    var[1] = 2;
    free(var);
    free(var);
}
```

When does this happen in real-world scenarios?
Memory Errors

• Freeing non-heap memory

```c
int main()
{
    int var[2]
    var[0] = 0;
    var[1] = 2;
    free(var);
}
```

When does this happen in real-world scenarios?
Memory Errors

• NULL pointer read / write
  
  ```c
  int main()
  {
    char *str = NULL;
    printf(str);
    str[0] = 'H';
  }
  ```

• NULL is never a valid location to read from or write to, and accessing them results in a “segmentation fault”
  – .... remember those memory segments?

When does this happen in real-world scenarios?
Memory Errors

• Uninitialized memory read

```c
int main()
{
    int *arr = malloc(sizeof(int)*10);
    int V2=arr[3];
}
```

When does this happen in real-world scenarios?
Memory error in action

```c
fawcett:error childs$ cat t.c
#include <stdio.h>

int main()
{
    int *X = NULL;
    printf("X is %d\n", *X);
}
fawcett:error childs$ gcc t.c
fawcett:error childs$ ./a.out
Segmentation fault
fawcett:error childs$  
```
Misc. Stuff for 4A
memcpy

MEMCPY(3)          BSD Library Functions Manual          MEMCPY(3)

NAME
    memcpy -- copy memory area

LIBRARY
    Standard C Library (libc, -lc)

SYNOPSIS
    #include <string.h>

    void *
    memcpy(void *restrict dst, const void *restrict src, size_t n);

DESCRIPTION
    The memcpy() function copies n bytes from memory area src to memory area
dst. If dst and src overlap, behavior is undefined. Applications in
which dst and src might overlap should use memmove(3) instead.

RETURN VALUES
    The memcpy() function returns the original value of dst.
sscanf

• like printf, but it parses from a string

    sscanf(str, "%s\n%d %d\n%d\n", magicNum, &width, &height, &maxval);

on:

    str=“P6\n1000 1000\n255\n”;

gives:

    magicNum = “P6”, width = 1000, height = 1000, maxval = 255
if-then-else

int val = (X < 2 ? X : 2);

\[
\begin{align*}
\leftrightarrow \\
\text{if } (X < 2) \\
\{ \\
\quad \text{val} = X; \\
\} \\
\text{else} \\
\{ \\
\quad \text{val} = 2; \\
\}
\end{align*}
\]