Lecture 5: memory errors
Announcements

• Projects
  – 1C delayed until Thurs the 14th
  – 4A assigned today
  – 2B assigned on Friday

• Discussion this week on debuggers/memory checkers

• No class on Weds April 27
  – likely a short YouTube replacement lecture
Outline

• Review
• Project 1C Overview
• Tar
• Memory Errors
• Project 4A
• Background Info for 4A
Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A
Build: The Actors

- File types
  - Source code
  - Object code
  - Executable code

- Programs
  - Compiler
  - Linker
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

```c
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

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

```c
fawcett:330 child$s$ cat t1.c
int doubler(int x)
{
    return 2*x;
}
fawcett:330 child$s$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 child$s$ gcc -c t1
fawcett:330 child$s$ gcc -c t2
fawcett:330 child$s$ gcc -o both
fawcett:330 child$s$ ./both
fawcett:330 child$s$ 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>”
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
Making a static library

Note the ‘#’ is the comment character
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/
C02LN00GFD58:multiplier hank$ 
```

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

```c
#include <multiplier.h>
#include <stdio.h>
int main()
{
    printf("Twice 6 is %d, triple 6 is %d\n", doubler(6), tripler(6));
}
```

- 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 computation

Basic idea: all details for compilation are captured in a configuration file ... you just invoke “make” from a shell
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$
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$
Outline

• Review
• Project 1C Overview
• Tar
• Memory Errors
• Project 4A
• Background Info for 4A
CIS 330: Project #1C
Assigned: April 7th, 2016
Due April 12th, 2016
(which means submitted by 6am on April 13th, 2016)
Worth 2% of your grade

Assignment: Download the file “Proj1C.tar”. This file contains a C-based project. You will build a Makefile for the project, and also extend the project.
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
Outline

• Review
• Project 1C Overview
• Tar
• Memory Errors
• Project 4A
• Background Info for 4A
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.
Unix command: tar

• `tar cvf 330.tar file1 file2 file3`
  – puts 3 files (file1, file2, file3) into a new file called 330.tar
• `scp 330.tar @ix:~`
• `ssh ix`
• `tar xvf 330.tar`
• `ls`
  
  file1 file2 file
Outline

• Review
• Project 1C Overview
• Tar
  • Memory Errors
• Project 4A
• Background Info for 4A
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
#include <stdio.h>

int main()
{
    int *X = NULL;
    printf("X is %d\n", *X);
}
```

```
fawcett:~ error childs$ cat t.c
fawcett:~ error childs$ gcc t.c
fawcett:~ error childs$ ./a.out
Segmentation fault
```
Outline

• Review
• Project 1C Overview
• Tar
• Memory Errors
• Project 4A
• Background Info for 4A
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 next week
Outline

• Review
• Project 1C Overview
• Tar
• Memory Errors
• Project 4A
• Background Info for 4A
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);
}
```
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+strlen("hello "));
    strncpy(str3, str, strlen("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

<table>
<thead>
<tr>
<th>Category</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copying</td>
<td><code>memcpy</code></td>
<td>Copy block of memory</td>
</tr>
<tr>
<td></td>
<td><code>memmove</code></td>
<td>Move block of memory</td>
</tr>
<tr>
<td></td>
<td><code>strcpy</code></td>
<td>Copy string</td>
</tr>
<tr>
<td></td>
<td><code>strncpy</code></td>
<td>Copy characters from string</td>
</tr>
<tr>
<td>Concatenation</td>
<td><code>strcat</code></td>
<td>Concatenate strings</td>
</tr>
<tr>
<td></td>
<td><code>strncat</code></td>
<td>Append characters from string</td>
</tr>
<tr>
<td>Comparison</td>
<td><code>memcmp</code></td>
<td>Compare two blocks of memory</td>
</tr>
<tr>
<td></td>
<td><code>strcmp</code></td>
<td>Compare two strings</td>
</tr>
<tr>
<td></td>
<td><code>strcoll</code></td>
<td>Compare two strings using locale</td>
</tr>
<tr>
<td></td>
<td><code>strncmp</code></td>
<td>Compare characters of two strings</td>
</tr>
<tr>
<td></td>
<td><code>strxfrm</code></td>
<td>Transform string using locale</td>
</tr>
<tr>
<td>Searching</td>
<td><code>memchr</code></td>
<td>Locate character in block of memory</td>
</tr>
<tr>
<td></td>
<td><code>strchr</code></td>
<td>Locate first occurrence of character in string</td>
</tr>
<tr>
<td></td>
<td><code>strcspn</code></td>
<td>Get span until character in string</td>
</tr>
<tr>
<td></td>
<td><code>strpbrk</code></td>
<td>Locate characters in string</td>
</tr>
<tr>
<td></td>
<td><code>strrchr</code></td>
<td>Locate last occurrence of character in string</td>
</tr>
<tr>
<td></td>
<td><code>strspn</code></td>
<td>Get span of character set in string</td>
</tr>
<tr>
<td></td>
<td><code>strstr</code></td>
<td>Locate substring</td>
</tr>
<tr>
<td></td>
<td><code>strtok</code></td>
<td>Split string into tokens</td>
</tr>
<tr>
<td>Other</td>
<td><code>memset</code></td>
<td>Fill block of memory</td>
</tr>
<tr>
<td></td>
<td><code>strerror</code></td>
<td>Get pointer to error message string</td>
</tr>
<tr>
<td></td>
<td><code>strlen</code></td>
<td>Get string length</td>
</tr>
</tbody>
</table>

## Macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NULL</code></td>
<td>Null pointer</td>
</tr>
</tbody>
</table>

## Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size_t</code></td>
<td>Unsigned integral type</td>
</tr>
</tbody>
</table>
memcpy

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.

I mostly use C++, and I still use memcpy all the time
sscanf

• like printf, but it parses from a string

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

on:

```c
str="P6\n1000 1000\n255\n";
```

gives:

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

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

if (X<2) {
    val = X;
}
else {
    val = 2;
}
```