CIS 190: C/C++ Programming

Lecture 3
Memory Management in C
Any Questions?
Outline

• (from last class) Testing
• Memory allocation
• Memory errors
• Errors
• Debugging
• Homeworks
Testing

• unit testing
  – literal tests to make sure code works as intended
  – e.g., `TwoPlusTwoEqualFour(...)` for an `Addition()` function

• edge case testing (or corner case, etc.)
  – ensure that code performs correctly with all (or at least many) possible input values
  – e.g., prevent program from accepting invalid input
/* get month from user in integer form */
printf("Please enter month: ");
scanf("%d", &month);
Simple Testing Example

/* get month from user in integer form */
printf("Please enter month: ");
scanf("%d", &month);
while (month < JAN_INT || month > DEC_INT)
{
    scanf("%d", &month);
}

Simple Testing Example

/* get month from user in integer form */
printf("Please enter month: ");
scanf("%d", &month);
while (month < JAN_INT || month > DEC_INT)
{
    printf("\n%d is an invalid month", month);
    printf("please enter between %d and %d:",
        JAN_INT, DEC_INT);
    scanf("%d", &month);
}
/ * print string up to number given by length (or full string, whichever is reached first) */

void PrintToLength(char str[],
                   int   length)
{
    int i;
    for (i = 0; i < length; i++)
    {
        printf(“%c”, str[i]);
    }
}
Common Edge Cases

• C-style string
  – empty string
  – pointer to NULL
  – without the \0 terminator

• Integer
  – zero
  – negative/positive
  – below/above the min/max
Outline

• (from last class) Testing
• Memory allocation
• Memory errors
• Errors
• Debugging
• Homeworks
Memory

- each process gets its own memory chunk, or *address space*

```
0x00000000
-----------------
Stack
Function calls, locals

4 GB address space
-----------------
Heap
Dynamically allocated memory

Global/static vars
"data segment"

Code
"code segment"
```
Memory

• each process gets its own memory chunk, or *address space*
Stack Allocation

• memory allocated by the program as it runs
  – local variables
  – function calls

• fixed at compile time
Heap Allocation

• dynamic memory allocation
  – memory allocated at run-time

• two options for allocating memory:
  – malloc()
  – calloc()
  • both require `#include <stdlib.h>` to work
malloc()

void* malloc ( <size to be allocated> )

char *letters;
letters = (char*) malloc(userVariable * sizeof(char));

• malloc returns a pointer to a contiguous block memory of the size requested
calloc()

void* calloc ( <number of elements>,
<size of type> )

float *grades;
grades = (float*) calloc(userVariable,
sizeof(float));

• calloc works very similarly to malloc, but it initializes all the allocated bits to zero
  – takes longer than malloc, so only use if needed
Casting Allocated Memory

- both `calloc()` and `malloc()` return a pointer of type `void`, so you must cast the memory to match the given type

```c
letters = (char*) malloc(userVariable * sizeof(char));

grades = (float*) calloc(userVariable, sizeof(float));
```
Casting Allocated Memory

• both `calloc()` and `malloc()` return a pointer of type `void`, so you must cast the memory to match the given type

```c
letters = (char*) malloc(sizeof(char) * userVariable);

grades = (float*) calloc(userVariable, sizeof(float));
```
Handling Allocated Memory

- **IMPORTANT**: before using allocated memory make sure it’s *actually been allocated*

- if memory wasn’t correctly allocated, the address that is returned will be **null**
  - this means there isn’t a contiguous block of memory large enough to handle request
Exiting in Case of NULL

• if the address returned is `null`, your program should exit
  – `exit()` takes an integer value
  – non-zero values are used as error codes

```c
if (grades == NULL) {
    printf("Memory not allocated, exiting.\n");
    exit(-1);
}
```
Managing Your Memory

• **stack** allocated memory is automatically freed when functions **return**
  – including **main()**

• memory on the **heap** was allocated by you – so it must also be freed by you
Freeing Memory

• done using the `free()` function
  – free takes a pointer as an argument:
    ```c
    free(grades);
    free(letters);
    ```

• `free()` does not work recursively
  – for each individual allocation, there must be an
    individual call to free that allocated memory
  – called in a sensible order
Freeing in Order

In what order would you free the nodes of this linked list?
Freeing in Order

In what order would you free the nodes of this binary tree?
Outline

• (from last class) Testing
• Memory allocation
• Memory errors
• Errors
• Debugging
• Homeworks
Memory Errors

• when we dynamically allocate memory, we are handling it directly
• have to be aware of possible errors like:
  – accessing off-limits memory
  – “losing” memory
  – running out of memory
  • not common nowadays, except in some embedded systems
Memory Leaks

• **memory leaks** occur when data is continually dynamically allocated but not freed

• access to the memory is then “lost”
  – for example, a loop that re-allocates memory to the same variable without freeing

• eventually we will run out of memory, and the program will crash or forcefully exit
for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}

Heap

arr ➔ ?
Memory Leak Example

```c
for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}
```

```
arr = (int*) malloc(...)
```

```
Heap
```

```
i = 0
```
for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}

i = 1
Memory Leak Example

```
for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}
```

```
i = 2
```

```
arr = (int*) malloc(...)  
arr = (int*) malloc(...)  
arr = (int*) malloc(...)  
```

```
arr
```
Memory Leak Example

for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}

i = 3
Memory Leak Example

for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}

i = 4
Memory Leak Example

```c
for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}
```

```
arr = (int*) malloc(...)
arr = (int*) malloc(...)
arr = (int*) malloc(...)
arr = (int*) malloc(...)
arr = (int*) malloc(...)
arr = (int*) malloc(...)
```

```
i = 5
```
Memory Leak Example

```c
for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}
```

![Diagram of memory allocation and deallocation for a memory leak example.](image-url)
Memory Leak Example

for (i = 0; i < var; i++) {
    arr = (int*) malloc(NUM * sizeof(int));
    /* check if arr == NULL */
}

i = 5
Mistakes When Using `free()`

• **double free**
  – freeing one pointer twice
    – without reallocating memory in-between frees
  – can cause a segfault

• **dangling pointer**
  – a pointer that points to freed memory
  – trying to access can cause a segfault
Segmentation Faults

- **segmentation faults** occur when you try to access memory that is off-limits

- segfaults occur during a program’s runtime
  – this can make them difficult to debug
Common Causes of Segfaults

• accessing out-of-bounds on an array

• accessing the memory address of uninitialized pointers

• accessing a pointer whose address points to memory that has been freed
C Trying to Be “Nice”

• when it can, C will do its best to shield you from errors like
  – freeing memory twice
  – accessing freed memory
  – manipulating freed memory

• but not
  – using uninitialized memory
C Being Nice

• double free memory
  – C will let it silently fail (most of the time)

• accessing freed memory
  – C will let you do this (most of the time)
  – BUT....
Killing with Kindness

• the data that was stored there has degraded or been corrupted when it was freed

• if code is changed so that freed memory is overwritten by a new “legitimate” allocation
  – you will suddenly have errors
  – that aren’t caused by the new code
  – makes it very difficult to debug
Outline

• (from last class) Testing
• Memory allocation
• Memory errors
• Errors
• Debugging
• Homeworks
Understanding Errors

hw2.c:87:7: error: 'foo' undeclared
Understanding Errors

\texttt{hw2.c:87:7: error: `foo` undeclared}

file in which error occurs
Understanding Errors

hw2.c:87:7: error: ‘foo’ undeclared

file in which error occurs

line number
Understanding Errors

`hw2.c:87:7: error: 'foo' undeclared`
Understanding Errors

file in which error occurs

character number

line number

degree of severity ‘error’ or ‘warning’

hw2.c:87:7: error: ‘foo’ undeclared
Understanding Errors

hw2.c:87:7: error: 'foo' undeclared

- file in which error occurs
- line number
- character number
- error message
- degree of severity: 'error' or 'warning'
#1 Rule of Debugging

- start with the **very first** error or warning
- recompile every time an error is fixed
  - errors will cascade
  - and de-cascade when fixed!
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;
```
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;

> gcc -Wall average.c
```
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}

avg = total/numStudents;

> gcc -Wall average.c

• the -Wall flag shows all of warnings
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;
```

```
> gcc -Wall average.c
average.c:5:5: warning: unused variable 'numStudnts'
average.c:22:17: error: 'numStudents' undeclared
average.c:25:13: error: 'numStudents' undeclared
```
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;
```

> gcc -Wall average.c

average.c:5:5: warning: unused variable `numStudnts'
average.c:22:17: error: `numStudents' undeclared
average.c:25:13: error: `numStudents' undeclared
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;
```

```
> gcc -Wall average.c
average.c:5:5: warning: unused variable `numStudnts'
average.c:22:17: error: `numStudents' undeclared
average.c:25:13: error: `numStudents' undeclared
```
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}

avg = total/numStudents;
```
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;

> gcc -Wall average.c
```
Cascading Errors

```c
int numStudents;
for (i = 0; i < numStudents; i++) {
    total += grades[i];
}
avg = total/numStudents;
```

• got rid of all 3 errors!
When Errors Occur

• compile time
  – pretty easy (normally typos or simple mistakes)

• linking
  – slightly harder (could be easy, could require rethinking how your code is laid out)

• run time
  – UGH (often difficult to pinpoint, and sometimes hard to spot at all)
  – best bet is to use a debugger
Common Compiler Errors

hw2.c:87:7: error: ‘foo’ undeclared

• if `foo` is a **variable**:  
  – forgot to declare  
  – misspelled (on declaration or on use)

• if `foo` is a **function**:  
  – forgot to `#include` file containing the prototype  
  – misspelled (on declaration or on use)
Common Compiler Errors

hw2.c:37:6: warning: unused variable ‘bar’

• variable was declared but not used
  – normally because variable declaration has a typo
  – if you’re in the midst of writing code, this warning may be *temporarily* acceptable
    – haven’t had a chance to use the variable yet
Common Compiler Errors

hw2.c:54: warning: suggest parentheses around assignment used as truth value

• often a mistake inside a control statement
  – you meant to use == not =
  – (you want equivalency, not assignment)
Common Compiler Errors

hw2.c: 51: error: expected ‘;’ before ‘for’

• missing semicolon on previous line of code
• ‘for’ is simply the word directly following the missing semicolon
  – could be ‘int’ or ‘if’ or a variable name, etc
Common Linker Errors

hw4.o: In function ‘main’:
hw4.c:91: undefined reference to ‘Fxn’

• linker can’t find code for ‘Fxn’ in any .o file
  – forgot to link .o file
  – misspelled named of Fxn
  – parameter list is different
    – differences between prototype/definition/call
Common Linker Errors

/usr/lib64/gcc/[...]/crt1.o: In function `_start':
/home/[...]/start.S:119: undefined reference to main

– you compiled a file that does not contain a main()
– without using the -c flag to indicate separate compilation
ABSOLUTELY TERRIFYING ERROR

• (story time!)
ABSOlutely Terrifying Error

> gcc -Wall structs.c
In file included from /usr/include/stdio.h:33:0,
  from structs.c:6.
/usr/lib64/gcc/xenial-aarch64-linux-gnu/4.7/include/stdio.h:233:1: error: expected `=', `;', `:', `asm' or `__attribute__' before `typedef'
In file included from /usr/include/stdio.h:74:0,
  from structs.c:6.
/usr/include/libc.h:307:3: error: unknown type name `size_t'
/usr/include/libc.h:311:67: error: `size_t' undeclared here (not in a function)
/usr/include/libc.h:330:62: error: expected declaration specifiers or `...' before `size_t'
/usr/include/libc.h:348:5: error: expected declaration specifiers or `...' before `size_t'
/usr/include/libc.h:470:19: error expected `=', `;', `:', `asm' or `__attribute__' before `__IO_setbuf'
In file included from structs.c:60:
/usr/include/stdlib.h:319:35: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:325:47: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:357:20: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:344:10: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:386:44: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:390:45: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:665:11: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:669:8: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:679:6: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:709:15: error: expected `=', `;','=', `asm' or `__attribute__' before `freed'
/usr/include/stdlib.h:715:15: error: expected `=', `;','=', `asm' or `__attribute__' before `fwrite'
/usr/include/stdlib.h:737:15: error: expected `=', `;','=', `asm' or `__attribute__' before `freed_unlocked'
/usr/include/stdlib.h:739:15: error: expected `=', `;','=', `asm' or `__attribute__' before `fwrite_unlocked'
In file included from structs.c:90:
/usr/include/string.h:62:42: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:65:56: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:92:48: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:129:39: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:137:39: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:143:57: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:150:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strxfm'
In file included from structs.c:90:
/usr/include/string.h:165:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strxfm'
/usr/include/string.h:180:45: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:281:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strspn'
/usr/include/string.h:285:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strspn'
/usr/include/string.h:395:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strlen'
/usr/include/string.h:402:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strlen'
/usr/include/string.h:423:12: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:447:33: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:451:58: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:455:31: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:458:34: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:556:61: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:573:34: error: expected declaration specifiers or `...' before `size_t'
/usr/include/string.h:576:39: error: expected declaration specifiers or `...' before `size_t'
in file included from structs.c:110:
/usr/include/stdlib.h:319:15: error: expected `=', `;','=', `asm' or `__attribute__' before `ctype_set_mb_cur_max'
in file included from structs.c:110:
/usr/include/stdlib.h:331:14: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:361:4: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:465:22: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:467:22: error: expected declaration specifiers or `...' before `size_t'
/usr/include/stdlib.h:479:36: error: expected declaration specifiers or `...' before `size_t'
in file included from /usr/include/stdlib.h:491:0,
  from structs.c:111:
/usr/include/alocah:32:22: error: expected declaration specifiers or `...' before `size_t'
in file included from structs.c:110:
/usr/include/alocah:497:22: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:502:45: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:502:65: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:755:9: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:755:25: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:760:34: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:760:50: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:839:6: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:842:6: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:846:31: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:850:31: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:859:36: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:863:34: error: expected declaration specifiers or `...' before `size_t'
/usr/include/alocah:870:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strxfrm'
/usr/include/alocah:873:15: error: expected `=', `;','=', `asm' or `__attribute__' before `strxfrm'

68
ABSOLUTELY TERRIFYING ERROR

EASY TO FIX
Debugging Basics

• if the error’s not clear from just looking at the code, you can try:
  
  • inserting probe statements with printf
    – (but adding a printf might change your error!)
  • rubber duck debugging
  • Googling the error message
  • using a debugger
Outline

• (from last class) Testing
• Memory allocation
• Memory errors
• Errors
• Debugging
• Homeworks
Debuggers

• see what is going on “inside” the program
  – more powerful and accurate than printf() probes

• examine individual variables (value & address)
  – can change variable’s value on the fly

• step through code line by line
  – can skip blocks of code you don’t want to see
Using DDD (or GDB)

• must use the ‘-g’ flag when compiling

• open program for testing using command line:
  ddd a.out
gdb hw2

• GDB – Gnu Project Debugger (text based)
• DDD – Data Display Debugger (GUI based)
DDD Basics

• debugger allows you to:
  • add breakpoints to stop the program at specific points
  • use ‘print’ or ‘display’ to show values (or addresses) of variables
  • step through code line by line
DDD Tips

• File -> Open Source
  – choose a different file to look at (and to set breakpoints in)

• Source -> Reload Source
  – refresh the source you’re using after recompiling without losing any breakpoints or data displays

• FINISH
  – executes the current “frame”
  – will pause when it hits a return (outside of main)
DDD Livecoding

• DDD livecoding example was taken wholesale from the sample session on this page:


• site also has more information about DDD
Outline

• (from last class) Testing
• Memory allocation
• Memory errors
• Errors
• Debugging
• Homeworks
Homework 2

• due tomorrow night @ midnight

• if you haven’t started yet – do it NOW!
Homework 3

• Memory Diagrams

• write legibly

• double check your work

• due at BEGINNING of class, on paper
  – no late days for this homework!