Lab 2

C Debugging

Announcements

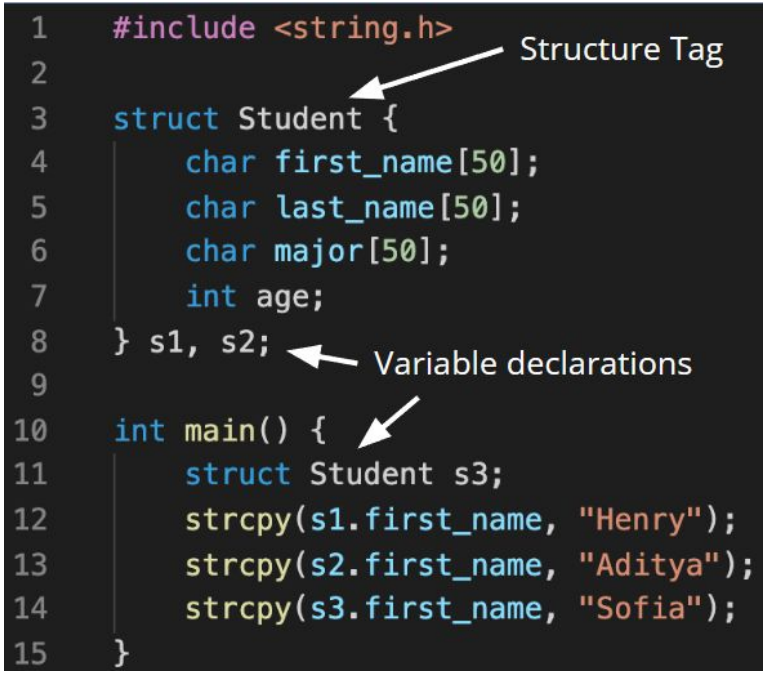
See assignment due dates below

Monday	Tuesday	Wednesday	Thursday	Friday
Lab 0 due		**HW 1 due**	**Lab 1 due** **Lab 2 due**	**Proj 1 due**

- Project 1: snek
  - Recommended to finish labs 1 + 2 before starting

Review: Structs

- Structs allow us to hold data items of different types in a single variable
- Structure Tag: optional, allows you to create new variables of this struct outside of the struct definition
- Two ways to access members
  - Use dot operator(.) with structs
  - Use arrow operator(->) with pointers to structs



Review: typedef

- Lets you avoid rewriting `struct` every time you want to declare a new struct variable
- Can no longer declare variables in the struct definition

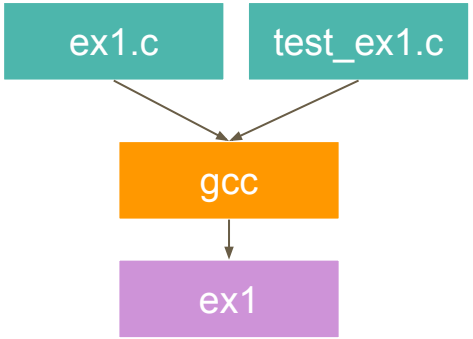
```c
#include <string.h>

typedef struct {
    char first_name[50];
    char last_name[50];
    char major[50];
    int age;
} Student;

int main() {
    Student s1, s2, s3;
    strcpy(s1.first_name, "Henry");
    strcpy(s2.first_name, "Aditya");
    strcpy(s3.first_name, "Sofia");
}
```

Review: Compiling/Running a C program

- `gcc -o ex1 ex1.c test_ex1.c`



- Can run the executable with `./ex1`

Compiler Warnings

- Generated to help you find potential bugs
- You should fix all warnings before trying to run code
  - However, not absolutely necessary

```
ex1_compiler_warnings.c: In function ‘make_course’:
ex1_compiler_warnings.c:13:22: warning: assignment to ‘char *’ from ‘char’ makes pointer fro
m integer without a cast [-Wint-conversion]
   13 |     new_course->name = *name;
      |                      ^
ex1_compiler_warnings.c:15:12: warning: returning ‘struct Course **’ from a function with in
compatible return type ‘struct Course *’ [-Wincompatible-pointer-types]
   15 |     return &new_course;
      |            ^~~~~~~~~~~
ex1_compiler_warnings.c:15:12: warning: function returns address of local variable [-Wreturn
-local-addr]
```

GDB (CGDB)

- GNU Project debugger
- Set breakpoints, run code, examine program state at specific execution
- cgdb ./pwd_checker

```
 3  #include "pwd_checker.h"
 4
 5  int main() {
 6      printf("Running tests...\n\n");
 7
 8      const char *test1_first = "Abraham";
 9      const char *test1_last = "Garcia";
10 ->   const char *test1_pwd = "qrtv?,mp!ltrA0b13rab4ham";
11      bool test1 = check_password(test1_first, test1_last, test1_pwd);
12      assert(test1);
13
14      const char *test2_first = "Anjali";
15      const char *test2_last = "Patel";
16      const char *test2_pwd = "Aj8r";
17      bool test2 = check_password(test2_first, test2_last, test2_pwd);
/home/uwuntu/labs/lab02/test_pwd_checker.c
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from [32mpwd_checker[m...
(gdb)
```

GDB commands

Command	Abbreviation	Description
start	N/A	begin running the program and stop at line 1 in main
step	s	execute the current line of code (this command will step into functions)
next	n	execute the current line of code (this command will **not** step into functions)
finish	fin	executes the remainder of the current function and returns to the calling function
print [arg]	p	prints the value of the argument
quit	q	exits gdb

GDB commands (cont.)

Command	Abbreviation	Description
break [line num or function name]	b	set a breakpoint at the specified location, use filename.c:linenum to set a breakpoint in a specific file
conditional break (ex: break 3 if n==4)	(ex: b 3 if n==4)	set a breakpoint at the specified location only if a given condition is met
run	r	execute the program until termination or reaching a breakpoint
continue	c	continues the execution of a program that was paused

Valgrind

- Tool which tracks your memory usage
- Can expose hard-to-track “heisenbugs”
- `valgrind ./executable_name`
  - `valgrind --leak-check=full ./executable`

```
==29797== HEAP SUMMARY:
==29797==     in use at exit: 8 bytes in 1 blocks
==29797==   total heap usage: 11 allocs, 10 frees, 1,061 bytes allocated
==29797==
==29797== LEAK SUMMARY:
==29797==    definitely lost: 8 bytes in 1 blocks
==29797==    indirectly lost: 0 bytes in 0 blocks
==29797==      possibly lost: 0 bytes in 0 blocks
==29797==    still reachable: 0 bytes in 0 blocks
==29797==         suppressed: 0 bytes in 0 blocks
==29797== Rerun with --leak-check=full to see details of leaked memory
```

Valgrind

- Make sure to free all allocated memory!
- Using `--leak-check=full`:

```
==32334== HEAP SUMMARY:
==32334==     in use at exit: 8 bytes in 1 blocks
==32334==   total heap usage: 11 allocs, 10 frees, 1,061 bytes allocated
==32334==
==32334== 8 bytes in 1 blocks are definitely lost in loss record 1 of 1
==32334==    at 0x4C31B0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==32334==    by 0x108784: alloc_str (in /home/cc/cs61c/fa22/staff/cs61c-tac/bork)
==32334==    by 0x10884E: concat (in /home/cc/cs61c/fa22/staff/cs61c-tac/bork)
==32334==    by 0x108A30: main (in /home/cc/cs61c/fa22/staff/cs61c-tac/bork)
==32334==
==32334== LEAK SUMMARY:
==32334==    definitely lost: 8 bytes in 1 blocks
==32334==    indirectly lost: 0 bytes in 0 blocks
==32334==      possibly lost: 0 bytes in 0 blocks
==32334==    still reachable: 0 bytes in 0 blocks
==32334==         suppressed: 0 bytes in 0 blocks
```

Stack trace!

Live Demo