

COMP 310

C review

Letao Chen
09/06/2019

Outline

1. VM/Environment Setup
2. `hello_world.c` data types, strings, if/else, functions, loops, header files, I/O
3. *Pointers
4. Command Line Arguments
5. Environment Variables
6. Compiling
7. Debugging gdb, memory leaks, seg faults
8. Questions + Networking Session

VM Setup

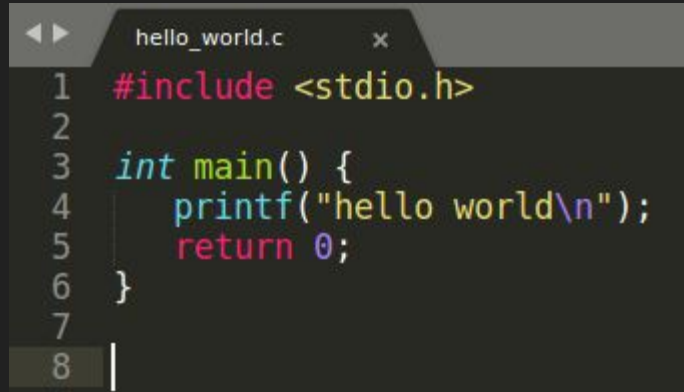
1. [VMWare Workstation \(Windows\)](#) / Fusion (Mac) OR [VirtualBox](#)
2. Recommended linux distro: [Ubuntu](#)
3. Code editor: Sublime, Visual Studio Code, Vim...

McGill Sci/Eng: [Download VMWare Pro Versions](#)

hello_world.c

stdio.h - I/O (printf)

\n - New line



```
hello_world.c x
1  #include <stdio.h>
2
3  int main() {
4      printf("hello world\n");
5      return 0;
6  }
7
8  |
```

hello_world.c

Example w/ user input

Can use `scanf` instead of `fgets`

```
hello_world_extended.c x
1  #include <stdio.h>
2
3  int main() {
4
5      // same as `char hello[6] = {'H', 'e', 'l', 'l', 'o', '\0'};`
6      // strings are null terminated
7      char hello[] = "Hello";
8
9      // lets get the user's name
10     char enterYourName[] = "Enter your name: ";
11     int userInputLen = 100;
12     char userInput[userInputLen];
13
14     printf("%s", enterYourName);
15     fgets(userInput, userInputLen, stdin);
16
17     // say hello
18     printf("\n%s %sWelcome to COMP 310\n", hello, userInput);
19
20     return 0;
21 }
```

hello_world.c

Functions

```
hello_world_extended.c x
1  #include <stdio.h>           // fgets, printf, stdin
2
3  // function declaration
4  void getUserName(char userInput[], int len);
5
6  int main() {
7      // lets get the user's name
8      int userInputLen = 100;
9      char userInput[userInputLen];
10     getUserName(userInput, userInputLen);
11
12     // say hello
13     printf("\nHello %sWelcome to COMP 310\n", userInput);
14
15     return 0;
16 }
17
18 void getUserName(char userInput[], int len) {
19     char enterYourName[] = "Enter your name: ";
20     printf("%s", enterYourName);
21     fgets(userInput, len, stdin);
22 }
```

hello_world.c

If/else example

Notice C does not have boolean type

{1: true, 0: false}

```
if_else.c x
1  #include <stdio.h> // printf
2  #include <stdlib.h> // srand, rand
3  #include <time.h> // time
4
5  int isEven(int n);
6
7  int main() {
8
9      srand(time(NULL)); // seed random number generator
10     int r = rand();
11
12     if (isEven(r)) {
13         printf("%i is even\n", r);
14     } else {
15         printf("%i is odd\n", r);
16     }
17
18     return 0;
19 }
20
21 int isEven(int n) {
22     return n % 2 == 0;
23 }
```

hello_world.c

Arrays, while and for loops

Notice no ``.length`` attribute

```
arrays.c x
1  #include <stdio.h>
2
3  int main() {
4
5      int yourComp310Marks[5] = { 34, 55, 49, 5, 21 };
6
7      int i = 0;
8      while (i < 5) {
9          printf("%i ", yourComp310Marks[i]);
10         i++;
11     }
12
13     printf("\n");
14     for (i = 0; i < 5; i++) {
15         printf("%i ", yourComp310Marks[i]);
16     }
17     printf("\n");
18
19     return 0;
20 }
```


*Pointers

`*` - dereference operator

`&` - memory location operator

Don't do:

`*pointer = &yourComp310Grade`

```
pointers.c
1  #include <stdio.h>
2
3  int main() {
4      int *pointer = NULL; // no value yet, good practice to NULL
5      int yourComp310Grade = 12;
6      int personSittingNextToYouGrade = 100;
7
8      // set the value of the ptr
9      // to the mem loc `&` of `yourComp310Grade`
10     // same as `int *pointer = &yourComp310Grade;`
11     pointer = &yourComp310Grade;
12     printf("My grade is %i\n", *pointer);    // `*` - derefs ptr
13                                           // 12
14
15     *pointer += 4;
16     printf("My grade is %i\n", yourComp310Grade);    // 16
17
18     pointer = &personSittingNextToYouGrade;
19     printf("Person beside me has %i\n", *pointer);    // 100
20
21     return 0;
22 }
```

*Pointers

To access arrays and 2d arrays, you can use ``array[i]`` and ``array[i][j]``

Here are last year's slides to demonstrate pointer notation for 2d arrays

Pointer to Pointer

- A pointer to pointer is a form of multiple indirection: a chain of pointers.



- The first pointer contains the address of the second pointer, which points to the location that contains the actual value.
- A variable that is a pointer to a pointer must be declared as such, i.e., placing an additional asterisk in front of its name.
 - `int ** variable;`

Slides by Farimah Poursafaei

2D Arrays/Pointers

- An array is a contiguous block of memory. A 2D array of integer of size m by n is defined as:

- `int A[m][n];`

- The number of byte necessary to hold A is **$m*n*\text{sizeof(int)}$** .
- How a 2D array is store?

A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]
---------	---------	---------	---------	---------	---------

- When a 1D array is declared as `int A[n]`, the name of the array A, is viewed as a pointer (const) to the block of memory where the array A is stored.
- We can access array elements using `[]` operator as `A[i]` or using pointer operator `*(A+i)`.
- A 2D array is viewed as an array of 1D array. That is, each row in a 2D array is a 1D array, Therefore given a 2D array A, `int A[m][n]`, we can think of `A[0]` as the address of row 0, `A[1]` as address of row 1 etc.

2D Array/Pointers – cont'd

- To find, for example $A[0][2]$, we do the following
 - $A[0][2] = *(A[0] + 2);$
- In general, $A[i][j] = *(A[i] + j);$
- Also, $A[0] = *A$, Therefore: $A[i][j] = *(A[i] + j) = (*(A + i) + j);$
- **Therefore, if A is a 2D array of integers, we can think of A as a pointer to pointer to an integer. That is $**$.**
- A dereference of A , of $*A$ gives the address of row 0 or $A[0]$ and $A[0]$ is an int^* .
- Dereference of $A[0]$ gives the first entry of A or $A[0][0]$, that is $**A = A[0][0]$.

Command Line Args & Env Vars

Taken from last year's slides

Careful: We do not have your Env Vars when grading

Command line arguments

- When a program is executed, it receives information about the context in which it was invoked in two ways:
 - *The first mechanism is through **Program Arguments**.*
- When executing your **C** programs, it is possible to pass some values from the command line; these values are called **command line arguments**.
- Why are they important?
 - *You can control your program from outside instead of hard coding the values inside the code.*
- How are the command line arguments handled?
 - *By using `main()` function arguments:*
- **argc**: refers to the number of arguments passed
- **argv[]**: a pointer array which points to each argument passed to the program

Command line arguments – cont'd

- ***arg[0]***: holds the name of the program itself
- ***arg[1]***: a pointer to the first command line argument
- ****arg[n]***: the last argument
- What is the value of ***argc*** if no arguments are supplied? What about having one arguments?
 - *All the command line arguments are passed separating by **space**.*
- `./a.out testArg1 testArg2`
- What if the argument itself has a space?
 - *Put them inside double quotes “”, or single quotes ‘’.*

Environment Variables

- When a program is executed, it receives information about the context in which it was invoked in two ways:
 - The second mechanism is through **Environment Variables**.
- The *environment* keeps track of information that is shared by many programs, changes infrequently, and that is less frequently used.
- Programs executed from the shell inherit all of the environment variables from the shell. The shell environment variable are set by using assignments and export command in the shell.
- Name of environment variables are *case-sensitive* and must not contain “=”.
- The value of an environment variable can be accessed with *getenv* function.
- Libraries should use *secure_getenv* instead of *getenv*, so that they do not accidentally use untrusted environment variables.
- Modifications of environment variables are not allowed in multi-threaded programs.

Environmental Variables – cont'd

- The environment variable define different things that may affect your program's behavior, like:
 - *The name of the default shell or text editor, the PATH that is search for binary executables,*
- Mostly, we see the *main* function signature as this:
 - *int main (int argc, char** argv[])*
- However, the signature more accurately looks like this:
 - *int main (int argc, char ** argv[], char **environ)*
 - *The third parameter is an array of KEY=VALUE.*
- You can get list of all environment variables by running *env* command.
- From a C program, you can get the value associated with an environment key by calling *getenv*:
 - *Char * getenv(const char *name)*
- You can set (or clear) the value associated with an environment variable by calling *putenv*:
 - *int putenv (char *string)*

Compiling

Stick w/ C standard libraries (unless specified)

- Do not import anything you do not need

If your code does not compile, try for making the marking easier

Avoid any compilation **errors and warnings**

If you compiled with any special flags, make a **visible comment**

Debugging

Compile w/ `-g` flag

```
lchen120@ubuntu:~/Documents/COMP310_TA$ gcc -g debugging.c
lchen120@ubuntu:~/Documents/COMP310_TA$ ./a.out
A1: 34% A2: 55% A3: 49% A4: 5% A5: 21%
Segmentation fault (core dumped)
lchen120@ubuntu:~/Documents/COMP310_TA$
```

```
pointers.c x debugging.c x
1 #include <stdio.h>
2
3 void printArrayWithWhileLoop(char *array[], int len);
4 void printArrayWithForLoop(char *array[], int len);
5
6 int main() {
7     int len = 5;
8     char *yourComp310Marks[5] =
9         { "A1: 34%", "A2: 55%", "A3: 49%", "A4: 5%", "A5: 21%" };
10
11     printArrayWithWhileLoop(yourComp310Marks, len);
12     printArrayWithForLoop(yourComp310Marks, len);
13
14     return 0;
15 }
16
17 void printArrayWithWhileLoop(char *array[], int len) {
18     int i = 0;
19     while (i < len) {
20         printf("%s ", array[i]);
21         i++;
22     }
23     printf("\n");
24 }
25
26 void printArrayWithForLoop(char *array[], int len) {
27     int i;
28     for (i = 0; i <= len; i++) {
29         printf("%s ", array[i]);
30     }
31     printf("\n");
32 }
```

Debugging

> where - show call stack

> up/down - go up/down stack

> list - shows snip of code

> print <var>

```
lchen120@ubuntu: ~/Documents/COMP310_TA
lchen120@ubuntu:~/Documents/COMP310_TA$ gdb a.out
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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 a.out...done.
(gdb) run
Starting program: /home/lchen120/Documents/COMP310_TA/a.out
A1: 34% A2: 55% A3: 49% A4: 5% A5: 21%

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7a5bcc0 in _IO_vfprintf_internal (s=0x7ffff7dd2620 <_IO_2_1_stdout_>,
    format=<optimized out>, ap=ap@entry=0x7ffff7dce8) at vfprintf.c:1632
1632 vfprintf.c: No such file or directory.
(gdb)
(gdb)
(gdb) where
#0 0x00007ffff7a5bcc0 in _IO_vfprintf_internal (s=0x7ffff7dd2620 <_IO_2_1_stdout_>,
    format=<optimized out>, ap=ap@entry=0x7ffff7dce8) at vfprintf.c:1632
#1 0x00007ffff7a62899 in __printf (format=<optimized out>) at printf.c:33
#2 0x0000000004006f4 in printArrayWithForLoop (array=0x7ffff7dfe00, len=5) at debugging.c:29
#3 0x00000000040063e in main () at debugging.c:12
(gdb) up
#1 0x00007ffff7a62899 in __printf (format=<optimized out>) at printf.c:33
33 printf.c: No such file or directory.
(gdb) up
#2 0x0000000004006f4 in printArrayWithForLoop (array=0x7ffff7dfe00, len=5) at debugging.c:29
29 printf("%s ", array[i]);
(gdb) print i
$1 = 5
(gdb) print array[i]
$2 = 0x823b5621e045db00 <error: Cannot access memory at address 0x823b5621e045db00>
(gdb) quit
A debugging session is active.

    Inferior 1 [process 3430] will be killed.

Quit anyway? (y or n) y
lchen120@ubuntu:~/Documents/COMP310_TA$
```


Debugging

> step - step in

> next - step over

> finish - step out

> continue

> break <breakpoint>

> delete <breakpoint>

> clear - clears all breakpoints

```
lchen120@ubuntu: ~/Documents/COMP310_TA
lchen120@ubuntu:~/Documents/COMP310_TA$ gdb a.out
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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 a.out...done.
(gdb)
(gdb) break debugging.c:29
Breakpoint 1 at 0x4006cb: file debugging.C, line 29.
(gdb) run
Starting program: /home/lchen120/Documents/COMP310_TA/a.out
A1: 34% A2: 55% A3: 49% A4: 5% A5: 21%

Breakpoint 1, printArrayWithForLoop (array=0x7fffffffde00, len=5) at debugging.c:29
29      printf("%s ", array[i]);
(gdb) next
28      for (i = 0; i <= len; i++) {
(gdb) next

Breakpoint 1, printArrayWithForLoop (array=0x7fffffffde00, len=5) at debugging.c:29
29      printf("%s ", array[i]);
(gdb) finish
Run till exit from #0  printArrayWithForLoop (array=0x7fffffffde00, len=5) at debugging.c:29

Breakpoint 1, printArrayWithForLoop (array=0x7fffffffde00, len=5) at debugging.c:29
29      printf("%s ", array[i]);
(gdb) print i
$1 = 2
(gdb) finish
Run till exit from #0  printArrayWithForLoop (array=0x7fffffffde00, len=5) at debugging.c:29

Breakpoint 1, printArrayWithForLoop (array=0x7fffffffde00, len=5) at debugging.c:29
29      printf("%s ", array[i]);
(gdb) clear
Deleted breakpoint 1
```

Questions + Networking

Discussion is encouraged. Absolutely no plagiarism, however.