Software Construction Laboratory

Week 4 Part 1

35L Lab 3

Mushi Zhou

Winter 2017

Prof. Paul Eggert

UCLA

Grades for Assignment 1

- Has been posted on MyUCLA
- The average is 90
- The high is 99
- See comments for grade breakdown
- Good job everyone!
- If you have questions, email me

Assignment 10

- Google Signup Form:
  https://https://docs.google.com/spreadsheets/d/1OGHR7sfjtyuCiMjxmZ4XgoGRChEbpcktY-DMAwTmCNY/edit#gid=0
- Deadline to signup is this Wednesday before class!
- You will lose 5 points for this assignment if I can’t approve your topic by the deadline
- Your entries will be highlighted green when approved, otherwise its orange and you need to modify it and then de-highlight so that I know you have modified
- Read comments for others so that you can better select your topic

Assignment 10

- Stick with your topic, contact me if need to change (Won’t be able to modify yourself)
- Show up and be on time
- If can’t make it, contact me in advance, otherwise -50% grades of presentation + a makeup

Grading of Assignment

- Report & Slides due one week after your presentation
- Late policy applies
- Grade = Presentation (39) + Slides(6)+ Report (55)
- Basic rubrics for both presentation and report can be found on CCLE
- Graded also based on the particular assignment requirements and interests

Tips for Presentation

- Keep 7-10 minutes
- Don’t dive too deep into the topic
- Keep a maximum of 10 slides
- Don’t put too much text on slides
- Try to allow everyone to understand what you are talking about
- Make it interesting & **relevant**
- Be prepared to answer questions if there are any
- Be prepared to ask questions to presenters since the staff could be on your final exam

Outline for This Week

- Introduction to C
- Difference between C and C++
- Pointers in C
- Dynamic memory allocations
- Debuggers (Part 2)
- Common debugging tools (Part 2)

Introduction to C

- The most widely used programming language of all times
- Developed in 1970s
- Imperative
- Static typed system
- C99 and C11 (newer version)
- Tutorials Point is a really good source for C function definitions.

Difference Between C and C++

- You can think it as a subset of C++ with some changes
- No classes at all
- No function overloading
- No function in structures
- No namespace
- Free functions
- Function driven
- Programmer can control memory explicitly

Struct in C

- typedef struct Database {

        int id_number;

        int age;

        float salary;

} database;

- database tmp;
- tmp.age = 10;
- struct Database tmp2;   // Same
- tmp2.age = 20;
- **Without typedef “database” is only a variable, not type**
- struct can contain other structs

Pointers in C

- int *Ptr;   //Declare Ptr as a pointer to integer
- int Var = 77;   // Define an integer variable
- Ptr = &Var;   //Let Ptr point to the variable Var
- (*Ptr) = 77;   // Accessing value of var

Dereferencing Pointers

double x, y, *ptr;	// Two double variables and a pointer to double.
ptr = &x;	// Let ptr point to x.
*ptr = 7.8;	// Assign the value 7.8 to the variable x.
*ptr *= 2.5;	// Multiply x by 2.5.
y = *ptr + 0.5;	// Assign y the result of the addition x + 0.5.

Pointer to Functions

```
double (*funcPtr)(double, double);
double result;
funcPtr = &pow;                 // Let funcPtr point to the function pow( )
                                // a built in function in c, power.
                                // The expression *funcPtr now yields the function pow( ).

result = (*funcPtr)( 1.5, 2.0 );          // Call the function referenced by funcPtr.
result = funcPtr( 1.5, 2.0 );             // The same function call.
```

- **You will need this for HW4**

void & bool

- C does not have a built in Boolean type
- Use #include <stdbool.h> if C99
- If C11 use

```
typedef int bool;
#define true 1        // #define is used to declare global constants
#define false 0       // Put this at the beginning of the file after include
```

- If there are no argument to a function, must put “void”, the same for functions returning nothing

```
void my_function (void) {};
```

Dynamic Memory Allocation

- malloc(size_t size)
  //allocates a block of memory whose size is at least *size*
- p = (int *) malloc (sizeof (int) * n);   // Allocates for an array of n integers
- free(p)   // frees the block of memory pointed to by p
                     // Always remember to free!
                     // Free the same pointer more than once will raise error
- calloc() is very similar to malloc(), but it initializes all fields to 0
- P = (int *) calloc(n, sizeof(int));
  // Allocates for an array of n integers initialized all to 0
- void *realloc(void *ptr, size_t new_size);
  // To adjust the allocated memory size
- **You will need this for HW4**

Opening & Closing Files

- FILE *fopen(const char *filename, const char *mode)
- // mode includes   w/r/a r+/w+/a+
- // return NULL if fails
- FILE *fp;   // file pointer
- int fclose(fp);   // Returns EOF if fails, otherwise 0

- Common Streams and their file pointers
- Standard input: stdin
- Standard output: stdout
- Standard error: stderr

Character I/O

- Reading/Writing characters
  - char c = getc(FILE **fp*);
  - putc(char c, FILE *pf);

    //get an unsigned char holding in an integer

- Reading/Writing Lines
  - char *fgets(char **str*, int *n*, FILE **stream*);

  // Stop if \n is read, n-1 char read or EOF is reached
  - int fputs(const char **s*, FILE * *stream*);

Formatted Input/Output

- Formatted Output
  - int fprintf(FILE * *fp*, const char * *format*, ...);
  - int fscanf(FILE * *fp*, const char * *format*, ...);

- Example:
  - int score = 120;
  - char player[ ] = "Mary";
  - fprintf(stdout, "%s has %d points.\n", player, score);
  - printf("%s has %d points.\n", player, score);

  -> Mary has 120 points.

Common Format Specifiers

Data Type	Format Specifier	Number of Bytes
char	%c	1
int	%d	4
long	%d	4
unsigned int	%u	4
short	%hi	2
float	%f	4
double	%f	8
Character array	%s	Many

Reading Numbers

- Byte vs Bits vs Digits
- 3251 is just decimal display of an integer
- Read 1 char != Read 1 digit != Read 1 byte
- Integer range from -2,147,483,648 to 2,147,483,647

- To read a number from stdin:
  - int inumber;
  - fscanf(stdin, “%d”, &inumber);

  -> 333333
  - inumber = 333333
  - This reads 4 bytes, not four digits, and that covers the above integer range

Sample Program

```
#include <stdio.h>
void printHelloWorld();

int main(int argc, char* argv[])          // Always need a main function of type int
{                                         // argv = input arguments to the program
   printHelloWorld();                     // argc = # of input arguments including the program name
   return 0;
}
void printHelloWorld()
{
   printf(“%s\n”, “Hello World!”);
}
```

Compiling

- gcc –o output -g example.c
- –g option indicates to include symbol and source-line info for debugging
- -o specifies the output filename
- -c can be used to generate object files (assembly file) without link them and allow multiple files to be linked together afterwards
- ./output will execute the program
- We will learn more about this in week 8