CME 211: Lecture 18

Topic:

- Programming in C
- What's next?

The C Programming Language

Why learn C?

- Mainly a subset of C++
- C is a low level language that has few abstractions over the hardware, so you get a better feel for how the hardware actually works
- Even within C++ programs, the most computationally intense parts should be very C like for efficiency
- In use for many applications: Linux kernel, CPython interpreter, low power or embedded systems, etc.

Where to learn?

21st Century C, Second Edition by Ben Klemens is a good place to start.

See: http://proquest.safaribooksonline.com/book/programming/c/9781491904428

C/C++ differences

C does not have:

- Support for Object Oriented Programming
- The C++ standard library
- Pass by reference or reference variables
- bool data type
- `new` / `delete` keywords replaced with `malloc` / `free` library functions
- `and` / `or` / `not` replaced with `&&` / `||` / `!`

Hello world

`src/hello.c`:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  /* Hello world program */
  printf("Hello world\n");
  return 0;
}
```

Output:

```
$ gcc src/hello.c -o src/hello
$ ./src/hello
Hello world
```

printf()

src/printf1.c:

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int a = 2;
  double b = 3.14;

  printf("a = %d\n", a);
  printf("b = %.1f\n", b);
  printf("argc = %d\n", argc);
  printf("argv[0] = %s\n", argv[0]);
  printf("a = %d, b = %f, argv[0] = %s\n", a, b, argv[0]);
  fflush(stdout);

  return 0;
}
```

Output:

```
$ gcc -Wall -Wextra -Wconversion src/printf1.c -o src/printf1
$ ./src/printf1
a = 2
b = 3.1
argc = 1
argv[0] = ./src/printf1
a = 2, b = 3.140000, argv[0] = ./src/printf1
```

Common mistake

src/printf2.c:

```c
#include <stdio.h>

int main()
{
  int a = 2;
  double b = 3.14;

  printf("a = %f\n", a);
  printf("b = %d\n", b);

  return 0;
}
```

Output:

```
$ gcc -Wall -Wextra -Wconversion src/printf2.c -o src/printf2
src/printf2.c: In function ‘main’:
```

```
src/printf2.c:8:10: warning: format ‘%f’ expects argument of type ‘double’, but argument 2 has type ‘in
   printf("a = %f\n", a);
          ^
src/printf2.c:9:10: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘doubl
   printf("b = %d\n", b);
          ^
$ ./src/printf2
a = 0.000000
b = 1
```

`for` loop, example 1

src/forloop1.c:

```
#include <stdio.h>

int main() {
  int n, sum;
  sum = 0;
  for (n = 0; n < 101; n++) {
    sum += n;
  }
  printf("sum = %d\n", sum);
  return 0;
}
```

Output:

```
$ gcc -Wall -Wextra -Wconversion src/forloop1.c -o src/forloop1
$ ./src/forloop1
sum = 5050
```

`for` loop, example 2

src/forloop2.c:

```
#include <stdio.h>

int main()
{
  unsigned int sum = 0;
  for (unsigned int n = 0; n < 101; n++)
  {
    sum += n;
  }
  printf("sum = %d\n", sum);

  return 0;
}
```

Output:

```
$ gcc -std=c89 -Wall -Wextra -Wconversion src/forloop2.c -o src/forloop2
src/forloop2.c: In function ‘main’:
src/forloop2.c:6:3: error: ‘for’ loop initial declarations are only allowed in C99 or C11 mode
   for (unsigned int n = 0; n < 101; n++)
```

```
   ^
src/forloop2.c:6:3: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 to compile your code
$ gcc -std=c99 -Wall -Wextra -Wconversion src/forloop2.c -o src/forloop2
$ ./src/forloop2
sum = 5050
```

Note: By default, on recent versions of Max OS X `gcc` is a wrapper around `clang`. `clang` seems more lenient with this issue when compiling with `-std=c89`. I installed `gcc-5` on my Mac and now get the expected behavior.

`gcc` standards support

- See `$ man gcc`
- See http://gcc.gnu.org/c99status.html

C89 vs C99

- In C89 you must declare all of your variables at the beginning of a scope block
- In C99 you can declare your variables anywhere (just like C++)
- In 2014, the Microsoft C compiler only supports C89

Writing to a file

`src/filewrite.c`:

```c
#include <stdio.h>

int main() {
  int n = 42;
  FILE *f = NULL; // Pointer to a file
  f = fopen("hello.txt", "w"); // Opens "hello.txt" for write
  if (f == NULL) // Make sure file was opened successfully
  {
    printf("Error: Failed to open file for write\n");
  }
  else
  {
    fprintf(f, "Hello\n");
    fprintf(f, "The answer is %d\n", n); // Print to file
    fprintf(f, "Goodbye\n");
    fclose(f); // close file
    f = NULL; // clear pointer
  }
  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/filewrite.c -o src/filewrite
$ ./src/filewrite
$ cat hello.txt
Hello
```

```
The answer is 42
Goodbye
```

C math library

src/math.c:

```c
#include <math.h>
#include <stdio.h>

int main()
{
  double a;

  a = sqrt(2.);
  printf("sqrt(2.) = %f\n", a);

  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/math.c -o src/math
$ ./src/math
sqrt(2.) = 1.414214
```

Functions

src/sum1.c:

```c
#include <stdio.h>

int sum(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int a = 2, b = 3, c;

  c = sum(a,b);
  printf("c = %d\n", c);

  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/sum1.c -o src/sum1
$ ./src/sum1
c = 5
```

Functions

src/sum2.c:

```c
#include <stdio.h>

int main()
{
  int a = 2, b = 3, c;

  c = sum(a,b); //Calling a function the compiler has no knowledge of

  printf("c = %d\n", c);

  return 0;
}

int sum(int a, int b)
{
  int c;
  c = a + b;
  return c;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/sum2.c -o src/sum2
src/sum2.c: In function ‘main’:
src/sum2.c:7:7: warning: implicit declaration of function ‘sum’ [-Wimplicit-function-declaration]
   c = sum(a,b); //Calling a function the compiler has no knowledge of
       ^
$ ./src/sum2
c = 5
$ g++ -std=c99 -Wall -Wextra -Wconversion src/sum2.c -o src/sum2
cc1plus: warning: command line option ‘-std=c99’ is valid for C/ObjC but not for C++
src/sum2.c: In function ‘int main()’:
src/sum2.c:7:14: error: ‘sum’ was not declared in this scope
   c = sum(a,b); //Calling a function the compiler has no knowledge of
               ^
```

Functions

src/sum3.c:

```c
#include <stdio.h>

int main()
{
  // change type to double
  double a = 2, b = 3, c;
  c = sum(a,b); // Calling a function the compiler has no knowledge of
  printf("c = %f\n", c);
  return 0;
}
```

```
double sum(double a, double b)
{
  double c;
  c = a + b;
  return c;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/sum3.c -o src/sum3
src/sum3.c: In function ‘main’:
src/sum3.c:7:7: warning: implicit declaration of function ‘sum’ [-Wimplicit-function-declaration]
   c = sum(a,b); // Calling a function the compiler has no knowledge of
       ^
src/sum3.c: At top level:
src/sum3.c:12:8: error: conflicting types for ‘sum’
 double sum(double a, double b)
        ^
src/sum3.c:7:7: note: previous implicit declaration of ‘sum’ was here
   c = sum(a,b); // Calling a function the compiler has no knowledge of
       ^
```

Type assumptions

C assumes unspecified types are integers

```
sum(a, b) {
  return a+b;
}
```

Here `a`, `b`, and the return type are all `int`.

Memory management

- `malloc()` allocates heap memory and returns a `void` pointer to the start of the allocation
- No guarantees about the state of initialization (i.e. the memory will have “random” data in it)
- `calloc()` works similar to `malloc()` except it initializes the memory to zero
- `free()` frees the memory allocated by `malloc()` or `calloc()`

Memory management

`src/malloc.c`:

```
#include <stdio.h>

#include <stdlib.h> // to get prototypes for malloc() and free()

int main() {

  // in C, initialize pointers to NULL
  double *a = NULL;

  // Use sizeof() function to get number of bytes required to store a double
```

```
  a = (double *)malloc(4*sizeof(double));

  a[0] = 0.; a[1] = 1.; a[2] = 2.; a[3] = 3.;

  printf("a[0] = %f\n", a[0]);
  printf("a[1] = %f\n", a[1]);
  printf("a[2] = %f\n", a[2]);
  printf("a[3] = %f\n", a[3]);

  // Free memory when done with it
  free(a);
  // clear out your pointer
  a = NULL;

  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/malloc.c -o src/malloc
$ ./src/malloc
a[0] = 0.000000
a[1] = 1.000000
a[2] = 2.000000
a[3] = 3.000000
```

Command line arguments

src/argv.c:

```
#include <stdio.h> // for printf
#include <stdlib.h> // for ato{i,f}

int main(int argc, char *argv[]) {
  if (argc < 4) return 1;
  char *file = argv[1];
  int n = atoi(argv[2]);
  double threshold = atof(argv[3]);
  printf("file = %s\n", file);
  printf("n = %d\n", n);
  printf("threshold = %f\n", threshold);
  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/argv.c -o src/argv
$ ./src/argv file.txt 4 3.7
file = file.txt
n = 4
threshold = 3.700000
```

Strings

- Strings in C are arrays of characters

- A null character, \0, denotes the end of the string (but not necessarily the end of the array)

```
char name[] = "Leland";
```

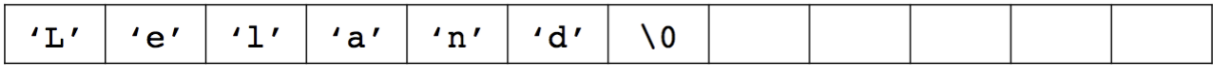

Figure 1: fig

char array

src/strings1.c:

```
#include <stdio.h>

int main()
{
  char name[] = "Leland";
  printf("name = %s\n", name);
  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/strings1.c -o src/strings1
$ ./src/strings1
name = Leland
```

Copying strings

src/strings2.c:

```
#include <stdio.h>

int main()
{
  char name[7];
  name = "Leland";
  printf("name = %s\n", name);
  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/strings2.c -o src/strings2
src/strings2.c: In function 'main':
src/strings2.c:6:8: error: assignment to expression with array type
   name = "Leland";
        ^
```

Copying strings

src/strings3.c:

```c
#include <stdio.h>
#include <string.h>

int main()
{
  char name[7];
  strcpy(name, "Leland");
  printf("name = %s\n", name);
  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/strings3.c -o src/strings3
```

Things to keep in mind:

- string copying in C can lead to several problems
- the function `strcpy` will keep copying until it hits the null character (`\0`). If there is no null character in the array, then the function may run on with out end, reading from and writing to parts of memory that it is not supposed to.
- C11 (the 2011 standard) introduces `strcpy_s`, which takes in a max size.

`fscanf()`

- For reading from files use the `fscanf()` function which is analogous to `fprintf()`
- `fscanf()` takes format specifiers for converting to the proper data type

```
float a;
...
fscanf(f, "%f\n", &a);
```

Reading numbers

`src/strings3.c`:

```c
#include <stdio.h>
#include <string.h>

int main()
{
  char name[7];
  strcpy(name, "Leland");
  printf("name = %s\n", name);
  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/strings3.c -o src/strings3
```

Reading numbers

`src/fileread.c`:

```c
#include <stdio.h>

int main()
{
  FILE *f = NULL;
  float a;

  char file[] = "numbers.txt";
  if ((f = fopen(file, "r")) == NULL)
  {
    printf("Error opening %s\n", file);
    return 0;
  }

  while(fscanf(f, "%f\n", &a) = EOF) {
    printf("%f\n", a);
  }

  fclose(f);
  f = NULL;

  return 0;
}
```

Output:

```
$ gcc -std=c99 -Wall -Wextra -Wconversion src/fileread.c -o src/fileread
src/fileread.c: In function ‘main’:
src/fileread.c:15:31: error: lvalue required as left operand of assignment
   while(fscanf(f, "%f\n", &a) = EOF) {
                               ^
$ cat src/numbers.txt
5.8
2.4
3.2
9.5
$ ./src/fileread
/bin/sh: ./src/fileread: No such file or directory
```

Pointers

If you use a Mac:

- Try Homebrew http://brew.sh/ to install software that does not come built in.
- Some people like MacPorts https://www.macports.org/, which does essentially the same thing. Seems like more people are using Homebrew these days.

If you use Windows:

- https://cygwin.com/: GNU and other tools to provide linux like functionality.
- http://www.mingw.org/: minimal gnu compiler toolchain for windows

For any operating system:

- Virtual machines (VirtualBox, VMware, Parallels) allow you to run another operating system concurrently.

- Vagrant https://www.vagrantup.com/ is a tool to create and configure lightweight, reproducible, and portable development environments. Uses VirtualBox under the hood.

Advice: Build software that works on Linux. If you are stuck on Windows or Mac, use the above tools to develop for Linux. As a result, you can run your code anywhere (personal computer, amazon cloud, your buddy's computer) without hardware or software licensing restrictions.

Where do you go from here?

- CME 212 - Advanced Programming for Scientists and Engineers
- CME 213 - Introduction to Parallel Computing using MPI, OpenMP, and CUDA
- CME 214 - Software Design in Modern Fortran for Scientists and Engineers
- CME 342 - Parallel Methods in Numerical Analysis

Course evaluations

- Please fill out the course evaluation in Axess
- I really do read them and make changes based on the feedback
- Evaluations due by December 15 at 8 am
- Can see your course grade as soon as they are entered, versus waiting until December 19