CAAM420: Daily Notes

Frank Portman

09/23/13

1 Compiling and Linking

Compilation is actually a two-step process. Only the first half is properly compilation. The first part turns human-readable text into machine code. The other part is that your text is never complete. You reference functions that you do not define within your source code. There is a declaration of square root in 'math.h' (also in cstd.h). The declaration is there but the definition is not. The definition, and in fact its translation into machine code, must be somewhere else. In order to put together a working application - the machine must find this other code and put it together with the output of the compiler. In our mysqrt.c example the sqrt definition was not found. There is a pile of machine code called the math library which contains a subset of the functions called in math.h. This last step is called linking (compilation and linking). The linking combines the output (translation) of the compiler with pre-compiled stuff somewhere else. In this case, the stuff is in a pile of code called a library.

The way to tell the linker where it is

```
gcc mysqrt.c -lm
```

The name of the linker is ld (ld returned 1 exit status. ld tried to run and couldn't find something) (ld is a program, linker,loader). We need to tell gcc (calls compiler and linker) where to find this additional code. We can access the math library with -lm (Why? Explained once we go over building our own libraries. Add the library called 'm'. -l says this is the name of a library).

The name of the linker is ld (which is also a program). We need to tell gcc (which calls the compiler and linker) where to find this additional code. We can access the math library with -lm. This will be explained more thoroughly when we go over building our own libraries. We add the library called 'm'. -l says this is the name of a library.

To call the mysqrt.c function we must use ./a.out with a floating point number.

```
./a.out 4.0
2.000
```

2 SCANF Family

To find out more about the SCANF family you can type:

```
man SCANF
```

We use fscanf converts string to other types. scanf is dual to printf. printf takes a number then converts to string and prints to terminal. scanf takes a string from terminal and converts it into whatever you tell it to (if its possible).

3 mysqrt2.c & Compiling Tips

The program doesn't expect an argument - it prompts you for one through scanf.

```
gcc mysqrt2.c -lm
./a.out
```

New idea: Let's say you're tired of typing ./a.out every time you want to run a program. Having to compile every time you want to execute a script because you wrote over a.out is annoying. The system provides a standard way to rename the compile link process.

```
gcc -o mysqrt2.c -lm (flag, anything with - means instruction) (-o means rename the output)
gcc -o fred mysqrt2.c -lm (rename it anything you want)
./fred
```

The ./ tells the runtime system where to look for the command. There are some places where it knows to look. There is an environment variable:

```
echo $PATH$
```

When you type a command, the system will look in all of those places. You won't see ./ in that list so the system won't look there unless you specify the path.

What if you want to have a bunch of arguments to a program? You can use the convention where you prompt each argument for its meaning. That can be bad if you don't want to see a bunch of stuff in your terminal and don't want to interact in the computer in this way.

4 Assigning Inputs

Another example: verify the addition of floating point doubles is commutative. Interestingly enough, addition is commutative. However, floating point numbers are not associative since intermediate results are rounded off.

```
#include "cstd.h"

int main(int argc, char ** argv) {

  double x = 0.0;
  double y = 0.0;
  double p;
  int i;
  for (i=1;i<argc;i++) {
    //    printf("%s\n",argv[i]);
    if (sscanf(argv[i],"x=%lf",&p)) x=p;
```

```
    if (sscanf(argv[i],"y=%lf",&p)) y=p;
    //    printf("x=%20.14e y=%20.14e\n",x,y);
  }
  printf("x=%20.14e y=%20.14e x+y=%20.14e\n",x,y,x+y);

}
```

We have two doubles, x and y as well as a buffer/workspace double p. Search through the arguments for something says x is equal to something else. It then converts the rest of the string as indicated by the program. sscanf returns the number of matches/conversions that it actually found. scanf is very literal - it looks for exactly 'x=' (no spaces).

Parameter parsing allows you to not have to manually test every case but is complex code to write.

5 Memory Issues in C

We have been using pointers quite a bit in these examples. Pointers are the source of an enormous amount of the power of C and its dependents. They also have the potential to cause harm.

The system software resides typically in the low addresses of the memory. If you can legitimately dereference a severely decremented pointer, you may very well be in the system memory area. Once you're there you can poke around and read passwords and cause all sorts of chaos.

```
ex_30_bus_error.c
gcc ex30_bus_error.c
./a.out
```

The system knows you shouldn't be doing this and kicks you out.

Segmentation fault is an attempt to access a memory address outside the segment of memory to which your program is localized. There's many ways in which you can attempt to access memory outside your segment. The most common way is to try to read or write off the end of an array (especially write). Declare an array of some length, and start writing past the end.

```
#include <stdio.h>

int main(int argc, char * argv[]) {

  int n;                  // max val of runaway index
  int i;                  // loop counter
  char s[10]="fruitcake"; // not enough for us!

  // we want to reference argv[1], so check that it's there...
  if (argc != 2) {
    printf("usage: ./a.out [int]\n");
    return 0;
  }

  // argv[0] is the name of the program, so argv[1] stores
  // the first argument
```

```
  printf("argv[0] = %s argv[1]= %s\n",argv[0],argv[1]);

  // but it's a string, even though it looks like an int -
  n=atoi(argv[1]);  // see p. 251 for atoi

  printf("assigning %d characters after %s\n",n,s);

  // make a really long string...
  s[9]=' ';
  for (i=0;i<n;i++) s[10+i]='z';
  s[10+n]='\0';

  printf("the result: %s\n",s);

  return 0;
}
```

The program assigns an array of length 10 with entries 'fruitcake'. It then takes another integer as an argument and assigns indexes that don't exist.

How can you capture the output of a program? Dump it to a file.

```
./a.out 1 >& junk
```

3 files are always open: stdin, stdout, stderr. It's the way system error messages get to you.

The core dump execution of the program results in hexadecimal outputs of certain files. The core dump is what was in each of the memory addresses associated to this program. It's unfortunate that this happened because it depends on how the program is loaded. Depending on your machine, you may be able to go more than 1 space over the array before the core dump. The memory segment boundary for your program can vary. The system boundary may not be at the end of your arrays. You may successfully be able to run many examples only to have the 32nd one blow up in your face. This can be troublesome if you share programs between machines. C has no intrinsic array type which accounts for some of the problem.

ex16_multid_array.c allocates a length 6 array but this is not the only way to grab a matrix in C. The most flexible way to grab a piece of memory to use as an array is by allocating some memory that is outside your program's stack. Use command 'malloc' (memory allocation) to allocate as much memory as you want up to the physical size of your core memory (and possible beyond in some cases). malloc returns an address in a part of memory called the heap. Stack is very well organized and neat. The heap is a pile of memory and whatever isn't active in the stacks is available for heap allocation. Read about dynamic memory allocation in K&R.

6 Fun Facts

Half a byte is called a nibble.