ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 12

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

20 February 2026

Announcements

- HW#3 was assigned (more on that in following slide)

HW#3 Update – Frequency?

- Should I break up homeworks more? They are a bit long.
- Might make sense to have 2 smaller homeworks a week so you aren't tempted to start a long complex one an hour before it's due

HW#3 Update – Relevance?

- Some people annoyed at how abstract they are.
- Yes, printing asterisk patterns seems sort of pointless but it's good practice in coding.
- The Labs in theory are there so you can actually have your code do real-world things (though you might find that's significantly harder)

HW#3 Help

- If you get stuck on a problem, skip ahead and come back to it
- Feel free to e-mail me with questions, I can log in and see your most recent attempt

HW#3 Annoyances

- I am using pre-existing questions from previous years
- Be sure to read the questions carefully. It is annoying they aren't consistent
- Often the problem is off-by-one, C makes that easy to do
- Sometimes they are whitespace issues
- Sometimes values are pre-initialized, sometimes they aren't

HW#3 Loop Review

- You are given a variable “n” which has a number of lines
- You also have variables “i” and “j” which you can use
- You want to print n lines, each line containing that many dollar signs. (The assumption is also probably you want to start at line 1 with 1, not line 0 with 0)
- If n=5, something like the following:

```
$
$$
$$$
$$$$
$$$$$
```

HW#3 Loop Review – for

```
for(i=1;i<=n;i++) {    // loop for number of lines
   for(j=0;j<n;j++) { // loop from 0 to line-1
      printf("$");    // print value
   }
   printf("\n");      // print newline before next line
}
```

HW#3 Loop Review – while

```
i=1;
while(i<=n) {             // loop for number of lines
   for(j=0;j<i;j++) {  // loop from 0 to line-1
      printf("$");       // print value
   }
   printf("\n");          // print newline before next line
   i++;
}
```

Lab #4 Preview

- Remember to bring laptop, breadboard, and USB-cable
- Loops
- Is going to look a lot like the loop questions from homework
- Going to output to the TFT/LCD display. Asterisk patterns

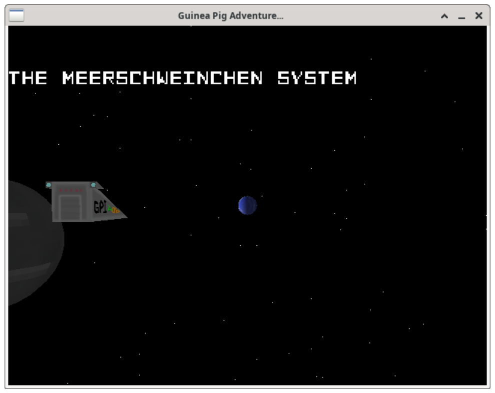
Questions about output more exciting than printf

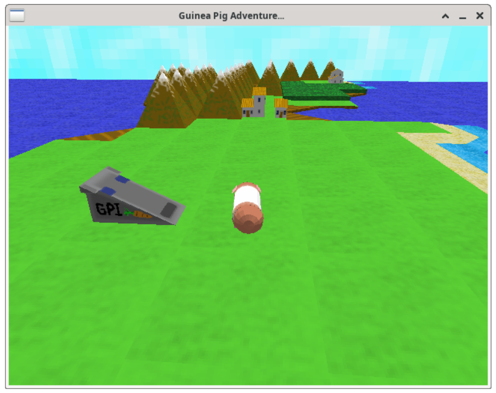
- As you know, most interfaces these days aren't text based at console
  Though weird old people like me often actually prefer things that way
- There are libraries you can use if you want to do things:
  - 2d-games: libSDL. A bit low level though. you might get a raw framebuffer where you set bits manually. If you want to draw circles/squares/text might be extra

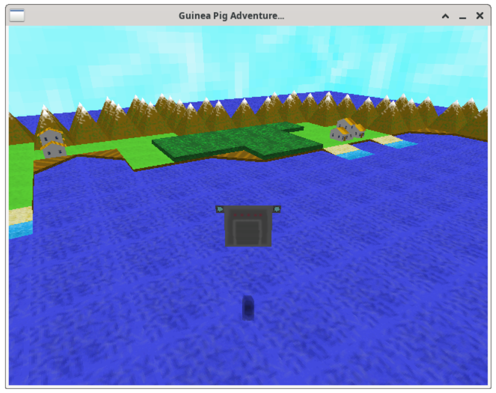
work

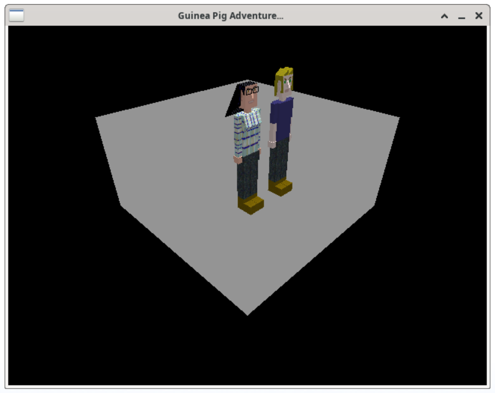
- GUI interfaces: things like gtk/qt. Can get buttons and stuff, and often a "canvas" sort of like a framebuffer
- Is there a C equivalent of pygame or maybe javascript canvas? Maybe?
- 3d: involves a lot of math. There are libraries like DirectX, OpenGL, or Vulkan. Again really low-level
- Most people making games use some sort of higher level infrastructure, like Unity, or Unreal, etc.

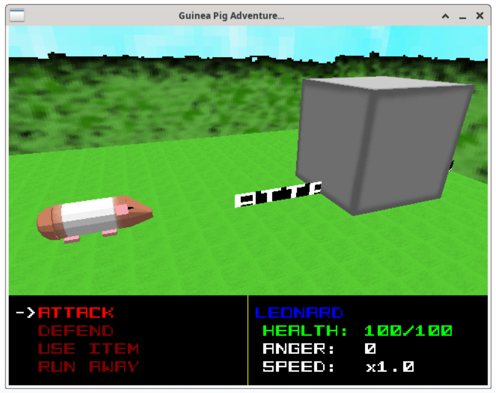
“Guinea Pig Adventure”: A 3D Game I made in OpenGL in 2001



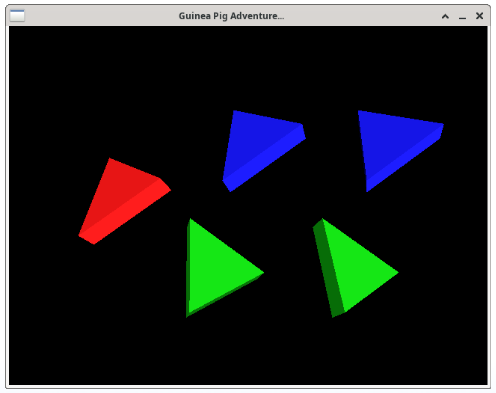








```
->ATTACK          LEONARD
  DEFEND           HEALTH: 100/100
  USE ITEM         ANGER:  0
  RUN AWAY         SPEED:  x1.0
```



Strings

- A string is how you store lines of text
- C has no dedicated string type
- A string in C is just an ASCII NUL (0) terminated array of characters

```
char st[10];  // string with room for 9
              // characters plus NUL at end
```

Strings in Other Languages

- Other, safer languages string data type includes the length
- How can you find out length in C?
  Loop looking for NUL

```
int strlen(char *string) {
  int length=0;
  while(string[length]!=0) length=length+1;
  return length;
}
```

String Declaration notes

- `char st[7]="Hello!";`

  `'H' 'e' 'l' 'l' 'o' '!' 0` in ASCII

  `0x48 0x65 0x6c 0x6c 0x6f 0x21 0x00`
- When using double quotes, C will automatically NUL terminate it for you
- Can automatically size a string by leaving off the size

  `char st2[]="Hello Again!";`
- Can also declare using pointers (more on that later)

Can you modify a string value?

- If you declare a string with a starting value, can you change it?
- Yes
- If you don't want this to happen you should declare it as a `const`

Poorly Sized String Initializations

- Show sample code that does this
- What happens if you try to declare too small

  `char st1[2]="Hello";` it will truncate it
- What if try one too small (no room for NUL)

  `char st2[5]="Hello";` it will leave NUL off (!)
- What happens if you give it much more room than needed?

  `char st3[100]="Hello";` it will pad with zeros

Arrays of Strings

- Can you do this? Yes

```
char colors[3][8]={"Red","Green","Blue"};
for(i=0;i<3;i++) printf("%s\n",colors[i]);
```

C Library String Routines

- There are various string manipulation routines in the C library
- You will need to include `#include <string.h>`

Getting Length of String

- `length=strlen(string);`
- Be careful: don't use `sizeof()` for this

  tells you the size of array which isn't necessarily the size of the string

Copying a String

- Use `strcpy(char *dest, char *src);`
- Note: you cannot just do `string="Hello";`

```
char string[100];
strcpy(string,"Hello");


;
```

Concatenating a String (copying to the end)

- Use `strcat(char *dest, char *src);`
- Note: you cannot just do `string=string+"Hello";`

```
char string[100]="Hello";
strcat(string,"World");


;
```

Comparing Strings

- Use `strcmp(char *string1, char *string2);`
- Note: you cannot just do `if (string1==string2)`
- You can imagine how you would do this yourself, have a loop and compare character by character
- Note! Unlike other C functions where “true” (nonzero) would mean success, for strcmp “0” means a match (TMI: this is because it compares by subtracting the strings)

```
char st1[100]="Hello",st2[100]="World";
```

```c
if (strcmp(st1,st2)==0) {
   printf("These strings are the same!\n");
}


;
```

String Manipulation Safety

- What if copy a string on top of one that's too big?
- What if you forget a NUL?

Aside: Buffer over-runs and C Security

- Aside on how underneath when you call a function, the return value is placed on a FIFO structure in memory called the "stack". Local variables also are. So if you write past the end of a string, you'll corrupt other variables and eventually over-write the return value. So when your function returns it will jump to the garbage address and crash your program
- Clever hackers can intentionally over-write return address to go to a location of their choice, and extra

clever hackers can overwrite memory with “shell-code” (Specially crafted machine code bytes) that is jumped to, and take over your computer

- This is why it is considered dangerous to use C code

Safer String Functions

- The big problem with strings is they can go off the end
- Better functions would take a length argument
- N variants: `strncpy()`, `strncat()`, `strncmp()`, etc

- What to do if copy is exact size of length?
  Truncating can cause unexpected issues
  Leaving NUL off bad
- These n variants do the "leave off NUL"
- There are l variants instead that do the "truncate"

- Ideally these functions would instead return an error if things didn't fit

Other Useful String Functions

- `strstr()` – search for a smaller string inside of a bigger one
- `strtok()` – split up string into tokens