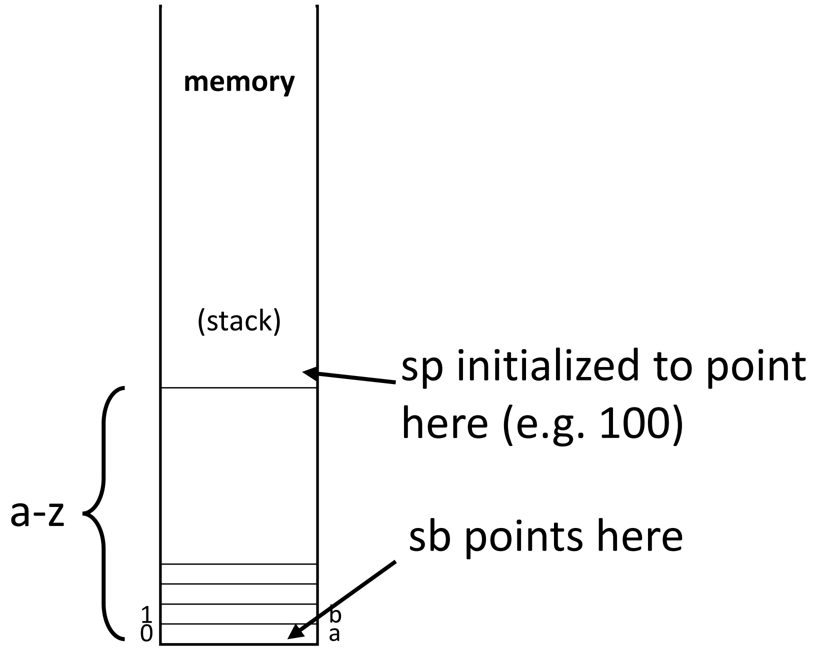
Tutorial 3:

Hints for A2 Task 3

(a) ... to generate code for nas (instead of sas)

```
a = 11;
print a + 22;
```

- Almost 1-1 mapping from sas to nas, except:
- nas does not have built-in named variables (a-z) which map directly to the a-z of the c4 language
- We map c4's a-z to the start of memory:
  - Use this same structure for Part (c) (Variables)
- Call this c4n (c4 that generates nas code), or if you use my c4.5, call it c4.5n



(b) Constants

```
int main() {
  char s[] = “hello”;
  long i;
  i = (long) &s;
  printf(“%s\n”, (char *) i);
}
```

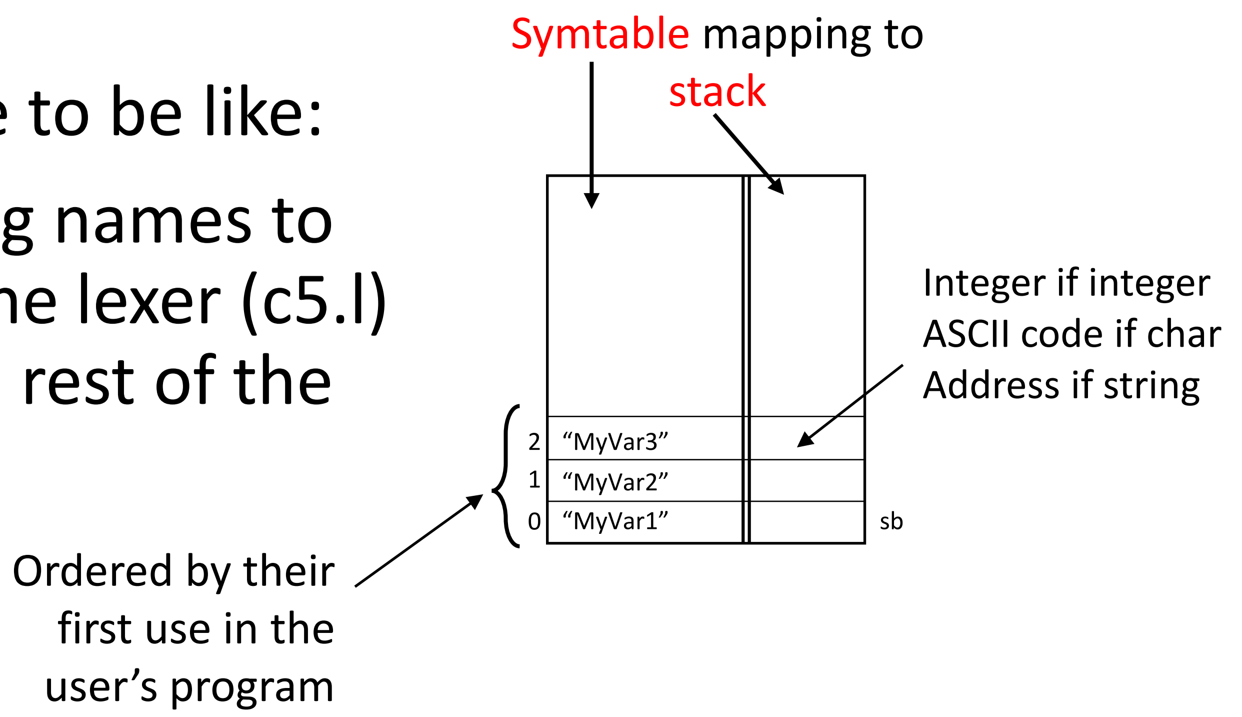
C has no complaint here

- Integer (already in c4) and char should be easy to do; in fact you could piggyback char on integer because char can be handled just like integer
- Note that nas’s stack/memory is actually an array of long (integers)
  - The reason: because we need to push memory addresses (for strings) onto it sometimes; memory addresses in most architectures require 8 bytes
  - You might have to use “cast” to print the stack top as integer or char
- So “string” has to be treated somewhat differently
  - The stack top is an address to a string stored in the heap
  - c5’s `a = “hello world”;` should translate to `push “hello world”; pop sb[0]`

Not the address; the actual string

(c) Variables

- Instead of a-z, we let users use long variable names of their own, e.g.: `ThisIsAVar123`
- I changed the symbol table to be like:
- I did all mappings from long names to stack locations (sb[…]) in the lexer (c5.l) so that I needn't touch the rest of the original code



(d) I/O statements

- Direct mapping. Easy.
- In fact, you only have to hand in c5c (parts (b)-(d)) and not c4n + c5c as the latter should include also the former

```
// FACTORIAL(x)

puts("This is factorial!");
ok = 0;
while (ok == 0) {
  puts_("Please input a number between 0 and 20: ");
  geti(input);
  if (input >= 0 && input <= 20) ok = 1;
}
temp = input;
fact = 1;
while (temp > 1) {
  fact = fact * temp;
  temp = temp - 1;
}
puts_("Cool! The factorial of "); puti_(input);
puts_(" is "); puti(fact);
```

(*) nas has problem with input when piped to

```
$ c5c fact.c5 | nas
```

✗

Ok, if your .c5 program doesn’t do input; the problem could be due to mixing up of stdin and yyin (defined by bison for reading the .c5 source) when yyin is also mapped to stdin. I tried to fix it but to no avail. Wanna try?

```
$ c5c fact.c5 >fact.nas
$ nas fact.nas
```

✓