ECS30: `char`, `printf`, `scanf`, overflow, random numbers …

Dr. Rob Gysel
1/13/17

`char` type

`ascii.c`

- `char` literal examples
  `'A'` `'0'` `'t'` `'\t'` (for tab)
- `char c = 'a';`
- `char` is generally 8 bits (1 byte)
- `char` is an 8-bit `int` converted via ASCII table
  (other encodings, ex UTF-8)
- Ex. 64th entry of ASCII
  `'@'` has `int` value
  `1000000` (2^8 = 64)

Dec	Hx	Oct	Html	Chr
64	40	100	@	@
65	41	101	A	A

Format Specifiers & `printf`/`scanf`

`typeformatspecifiers.c`

- Tells `printf` what type is being printed
  - `%d` for `int`
  - `%lf` for `double`
  - `%c` for `char`

```
    int dozen = 12;
    double pi = 3.14159265359;
    char lastVowel = 'u';
    printf("%d, %lf, %c\n", dozen, pi, lastVowel);
}
```

Format Specifiers & `printf`/`scanf`

`typeformatspecifiers.c`

- Tells `printf` what type is being printed
  - `%d` for `int`
  - `%lf` for `double`
  - `%c` for `char`

```
    int dozen = 12;
    double pi = 3.14159265359;
    char lastVowel = 'u';
    printf("%d, %lf, %c\n", dozen, pi, lastVowel);
}
```

- Order matters! 1st format specifier is for 1st variable listed, etc.

Format Specifiers & `printf`/`scanf`

`typeformatspecifiers.c`

- Tells `printf` what type is being printed
  `%d` for `int`
  `%lf` for `double`
  `%c` for `char`

```
    int dozen = 12;
    double pi = 3.14159265359;
    char lastVowel = 'u';
    printf("%d, %lf, %c\n", dozen, pi, lastVowel);
}
```

- Order matters! 1st format specifier is for 1st variable listed, etc.

Type casting & printf

`intprintf.c`

Probably not what you want:

```
int sixtyfour = 64;
printf("Without explicit type conversion:\n");
printf("sixtyfour as an int: %d\n", sixtyfour);
printf("sixtyfour as a char: %c\n", sixtyfour);
printf("sixtyfour as a double: %f\n", sixtyfour);
```

Type casting & printf

`intprintf.c`

Probably not what you want:

```
int sixtyfour = 64;
printf("Without explicit type conversion:\n");
printf("sixtyfour as an int: %d\n", sixtyfour);
printf("sixtyfour as a char: %c\n", sixtyfour);
printf("sixtyfour as a double: %f\n", sixtyfour);
```

Type casting & printf

`intprintf.c`

Probably not what you want:

```
int sixtyfour = 64;
printf("Without explicit type conversion:\n");
printf("sixtyfour as an int: %d\n", sixtyfour);
printf("sixtyfour as a char: %c\n", sixtyfour);
printf("sixtyfour as a double: %f\n", sixtyfour);
```

Asks `printf` to print an `int` as a `char`

Type casting & printf

`intprintf.c`

Probably not what you want:

```
int sixtyfour = 64;
printf("Without explicit type conversion:\n");
printf("sixtyfour as an int: %d\n", sixtyfour);
printf("sixtyfour as a char: %c\n", sixtyfour);
printf("sixtyfour as a double: %f\n", sixtyfour);
```

The correct way:

```
int nine = 9;
printf("With explicit type conversion:\n");
printf("nine as an int: %d\n", nine);
printf("nine as a char: %c\n", '0' + (char) nine);
printf("nine as a double: %f\n", (double) nine);
```

Type casting & printf

`intprintf.c`

What's happening on line 13?

`'0' + (char) nine`

1. `int` addition
   (implicit conversion)
2. Which `int`s?

Type casting & printf

`intprintf.c`

What's happening on line 13?

`'0' + (char) nine`

1. `int` addition
   (implicit conversion)
2. Which `int`s?

Dec	Hx	Oct	Html	Chr
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9

Type casting & printf

`intprintf.c`

What's happening on line 13?

`'0' + (char) nine`

1. `int` addition
   (implicit conversion)
2. Which `int`s?

`'0'` is `48`

`int nine = 9;`

`(char) nine` is `57`

And `57 - 48 = 9`

Dec	Hx	Oct	Html	Chr
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9

Representation of `ints` and `doubles`

- Internally, stored as sequences of bits (*binary digits*)
- `int`
  - All bits used to represent binary number
- `double`
  - 3 parts: sign (+/-), mantissa (significant digits), exponent
  - `double x =` (sign +/-) mantissa x 2^exponent

1 bit	52 bits	11 bits

Overflow `integeroverflow.c`

Numbers are represented with finite # of bits

- `int` has a min and max value.
  - Exceeding the range is an *integer overflow error*
- Floats only have a finite precision (some numbers are too small to represent accurately).
  - Wrong calculations due to finite precision are *numerical errors*
  - We won't discuss numerical error more in this class; deep topic

Overflow `integeroverflow.c`

Numbers are represented with finite # of bits

- `int` has a min and max value.
  - Exceeding the range is an *integer overflow error*
- Floats only have a finite precision (some numbers are too small to represent accurately).
  - Wrong calculations due to finite precision are *numerical errors*
  - We won't discuss numerical error more in this class; deep topic

Random Number Generation

- Random number generator `rand`
  - Not actually random: `rand` gets next number (between 0 and `RAND_MAX`) in a fixed sequence

`notrandom.c`

Random Number Generation

- Random number generator `rand`
  - Not actually random: `rand` gets next number (between 0 and `RAND_MAX`) in a fixed sequence
  - Sequence is determined by *seed value* sent to `srand`
  - Seed by `time` to get truly random numbers

`random.c`

Random Number Generation

Specifying the range of random numbers

1. Seed with `srand` and `time`
2. Use `rand` to generate numbers
3. Use mod to restrict range, e.g. numbers between 0 and 9:

   ```
   rand() % 10
   ```

   Between 1 and 10:

   ```
   1 + rand() % 10
   ```

Further Reading (optional)

- Article on integer overflow

References:

- `printf`, `scanf`
- `printf` format specifiers
- `scanf` format specifiers