APS 105 Lecture 5 Notes

Last lecture: basic arithmetic operators, * / + - () %
assignment operators, += -= *= %= /=
type casting and math library

Today: more on math library and random number generators

Recap:

```
int x = 5/3;    // x stores 1
double i = 5/3;    // i stores 1.0
double j = 5.0/3    // j stores 1.666...
double k = (double) 5/3    // k stores 1.66...
```

<math.h>
└ e.g. `printf("%lf", sqrt(4.0));`
`printf("%lf", sin(M_PI * 7));`

New:

```
double fmax (double x, double y)    //returns largest of x and y
double fmin (double x, double y)    //returns smallest of x and y
double floor (double x)    //returns largest int <= x
```

E.g.

```
int floorValue = floor(5.3);
printf("%d\n", floorValue);    → 5
```

floor(-5.3) // returns -6

double ceil (double x) //returns smallest int >= x

```
int ceilValue = ceil(-5.3);
printf("%d\n", ceilValue);  ⇒ prints -5
```

double fmod (double x, double y) //mod % for doubles

$\frac{5.3}{2.1} = 2.523$

printf("%lf", fmod(5.3, 2.1)); ⇒ prints 1.1

$0.523 * 2.1 = 1.1$

Very handy tool:

double rint (double x)

rounds to the nearest int, since it returns a double, it appends/converts/"type casts" the int to double to return it

printf("%lf", rint(-2.1)); ⇒ prints -2.0

For example,

Round a floating point number to the nearest 10th (1st decimal point)

Toy example: 2.18 ⟹ 2.2

① 2.18 ⇒ 21.8 (* 10)
② 21.8 ⇒ rint(21.8) ⇒ 22.0 (round to nearest int)
③ 22.0 ⇒ 2.2 (/10)

– DEMO onwards –

More Complex example:

Canada has no pennies

Round to the nearest nickle (5 cents)

1 nickle = 5 cents

1 dollar = 100 cents

Toy example: $2.94 ⇒ $2.95

$2.92 ⇒ $2.92

① How many nickles in $2.94?

2.94 * 100 / 5 = 58.8

2.94: double; *100: to pennies; /5: to nickles

② Round nickles to the nearest nickle

rint (58.8) ⇒ 59

③ So how many dollars is that?

59 * 5 / 100 = $2.95

*5: to pennies; /100: to dollars

printf("%lf", rint(price * 100 / 5) * 5 / 100);

Recall rint returns a double

Generating random numbers

↓

unpredictable number

#include <stdlib.h> → standard library

int rand()

returns int — takes no argument

↓

so it can produce any +ve int from 0 ~ $2^{31}-1$

2147483647

$2^{31}-1$ is a const defined as RAND_MAX

every time you call rand(), it produces a different #

```
printf("%d", rand());
printf("%d", rand());
printf("%d", rand());
```

every time you run the code, same set of random numbers are generated, is it really random then? NO.

It is a pseudo-random number generator.

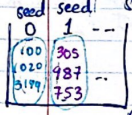
What if you want another set of random numbers?

You can change the set of random numbers by using a different "seed".

For example, every one in class write a set of 5 random numbers in chat.

Next time, I run my program, I will pick the set of numbers by student 1. In C, we pick the seed using

void srand( ) → by default the seed is 1
↑ unsigned int "seed"



But if I fix my seed, I will fall into using the same set of random numbers every time I run my code.

Solution: every time you run the code, choose another seed. But how?

Make seed depend on time!

#include <time.h>

time_t time (time_t * t)
function

no need to know these now

What you need to know is time (NULL)

returns time in UNIX:
# of seconds since Jan 1, 1970

so if we do srand (time (NULL))

our set of random numbers is different each time

You need to call srand( ) only once at the beginning of the code. If you call it before every rand(), it will return the same random number.

→ Can I play heads and tails using random numbers? Yes.

If random # is 0 → H
If random # is 1 → T

But rand() generates from 0 ~ $2^{31}-1$. How to limit the numbers?

Think about modulo % !!

0 % 5 = 0
1 % 5 = 1
2 % 5 = 2
3 % 5 = 3
4 % 5 = 4
5 % 5 = 0
6 % 5 = 1

(repeats again!)

% 5 will always return a # between 0 and 4
% 2 " " " " " " 0 and 1

so rand() % 2 → will produce a # 0 ~ 1

what if I want to produce a # 15 ~ 16

rand() % 2 +15 → 15 ~ 16