9/29/20 Note

Control structures and flow control

Boolean Algebra
- set mathematics on logical foundation
- Bool logic has true and false
- 3 basic operators
  - ∧ called and (& && in C)
  - ∨ disjunction → (|| in C)

And (conjunction)
- associative identities
- ~~DeMorgan's Law~~ distributive
- $A \wedge 1 = A$
- $A \wedge 0 = 0$
- $A \wedge A = A$

Or (inclusive / disjunction)
- $A \vee (B \vee C) = (A \vee B) \vee C$ · associative
- ~~DeMorgan's Law~~ distributive
- $A \vee 0 = A$
- $A \vee 1 = 1$
- $A \vee A = A$

Not (negation)
- DeMorgan:
  $\neg(A \wedge B) = \neg A \wedge \neg B$

Exclusive -or (XOR) first thing or second thing but not both
- $A \oplus B = (A \vee B) \wedge \neg(A \wedge B)$
- $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$
- Algebraic properties
  - $A \oplus A = 0$
  - $A \oplus 0 = A$
  - $A \oplus 1 = \neg A$
  - $A \oplus (B \oplus C) = (A \oplus B) \oplus C$

⊕ (XOR)	T	F
T	F	T
F	T	F

Logical Operators
< less
<= less than equal
== equal
> greater
>= greater than equal

True and False in C
- In C, zero (0) is false
- All that is not false is true
- logical ops. have type int
- You can have true and false if you:
  #include <stdbool.h>

if ()
- it affects next statement if Bool exp. is true
- even though { } are not required per a single statement, always use them
- Why?
  - it avoids error when adding statements

if () {...} else {...}

Nested if ()

if () {...} else if () {...} ...else {...}

Boolean Operators
&& = and
|| = or
~ = not

Short circuit evaluation
- false && anything is false
- true || anything is true
- stop evaluating as soon we know the result
- suppose we don't stop:
  - we would be dividing by zero
- we would follow a null pointer

```
switch (exp) {
    case 0:
        msg = "zero";
        break;
    case 1:
        msg
        break;
    default:
        msg
}
```

switch()
- allows you to select among a fixed set of alts
- b

equivalent goto code

What's wrong w/ goto?
- it's not about how you got there
- you could goto middle of
  - an if statement
  - switch statement or
  - a loop
- It is not even clear what that would mean.

What is a loop?
- repeats a seq. of code.
- progs. spend vast majority of their execution time in loops
- we will focus on loops: while, for, and do-while
- you can also create loops w/ the goto last day.

while()
- top-test loop.
  - test is evaluated before entering loop.
- executes statements as long as test condition remains true.

Equivalent goto code.
- you can implement it w/ goto statements
- not recommended.

for()
- also top test loop.
- three parts:
  - initialization
  - test, and
  - increment all together
- By convention they are related, but not required to be related.

equivalent while()
- This is the most general while statement
- the while statement is complete
  - which means you can implement any loop

do { } while
- bottom-test loop
- used when you want to perform the statement at least once
- continues to execute the enclosed statement as long as the Bool cond. remains true

Infinite loops
- execute forever
- the one you choose is a matter of style not of efficiency.
- How do you ever escape?
  - 'break;'

break
- immediately exits enclosing loop

Factorial example
- $n! = n * (n-1)!$

Asserts: you don't need to do factorials if.

When can I use goto?
- one place: non-local error handling
- this is when an exceptional condition that you cannot handle occurs
- It is not pretty.

Caution:
- You may have times when you want to skip remainder of a loop
- but please use sparingly

Let's compute $\sqrt{2}$!

- $\sqrt{2}$ is the same as: $x^2 - 2 = 0$
- we also know $0 \leq x \leq 2$,

Bisection Method

10/1/21 Notes

Word of Caution

- Function C are not the same in math functions
  - they may/not return a value C void returns nothing
  - may/may not have side-effects
- subroutine in other language
- In Java, it's a method

How is it different?

- math uses domain and range

- In C, we call function
  - function it returns value

function in programming:

- block of code that performs a certain task
- defined exactly once.
- must be declared before they are used.
- can declare & call a function as many times as desired.
- main()
  - is a special function.
  - Is run when program starts.
  - All other functions are subordinate main()

Why do we like functions?

- functions should:
  - define abstractions
  - give names to segs. of code
  - hide the implementation
- use them to:
  - refactor repeated segs. of code
  - simplify code to aid understanding
- Functions should never be:
  - Arbitrary segs. of statements

Function definition.
- function head
  - return type
    - define type of function's return value
    - return type could be void or any obj. type (except. array)
  - function_name()
  - parameters
    - contained in comma seperated list of decl.
    - if function has no parameters then void.
- Function block/body
  - declaration
    - declared variable inside a funct. bdy are local
  - assign new

Return values.
- funct. returns a value
- may be void
- return a struct (not recommended)
- return a pointer
- not an array

Function naming.
- same naming rules as variable
- can't start
  - start w/ a number or any punctuation other than _ and $
- use snake case
  - my_function_name

Parameters or Arguments
- If we have $f(x) = x \log(x+1)$ and we write $f(2)$ we substitute 2 for x and get $2 \log(2+1)$.
  - this is call-by-name and is rare.
  - C supports this as textual subs is macros and C Preprocessor.
- most prog-langs, use either call-by-name, call-by-reference or both.
- C uses call-by-value, except for arrays, and only b/c of their relation to pointers.

Parameters
- formal parameters
  - name of parameter as it is used in body
- Actual parameters
  - name of value that is passed to func
  - the value can be copied to the formal param.
  - or a ptr reference to the actual parameter
- Copy-in-Copy out means that should be copied, value is copied back out
  - C does not support this.

Call-by-value
- all functions use this.
- Arguments passed into a function are copied
  - any changes made to the parameters inside the function has no effect
- the called function copies the values in

Call-by-reference
- refs.

Tenary operator
```
h=(x<1) ? 1 : x;
if (x<1) {
return 1;
} else {
return x;
}
```

swap()
(does not have true call-by-reference so we use pointers.
- Addresses instead of values are passed to arguments

```
void swap (int *a, *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
    return;
}
int void main (void) {
    int x = 5;
    int y = 7;
    swap(&x, &y);
    printf("in! x = %d", x);
    printf("in! y = %d", y);
```

function prototype
- Syntax:
  return_type function_name(param);
- prototype must be declared either at beginning of program or in included header file.

#include
- a preprocessor directive,
- Before compiling C source files are processed by preprocessor
- preprocessing is a macro processor to transform programs before compiling
- macros in C operate through text replacement
- value/code of given file into current file.
- used to include functions defined in other libraries

#define
- A preprocessor directive that defines macros for the program
- C preprocessor performs all text replacements for defined macros prior to compilation

Conditional directives
- a set of preprocessor directives that use Cond. statements to include code selectively.
- #ifndef - checks statements when MACRO is defined

Header file
should only contain things that are shared b/t source files

#ifndef vs. #pragma once
- #ifndef checks macro will always work (use it more portable)

extern
- extends visibility of var and functions such that they can be called by any program file.

Static
- can be declared in and outside function.

10/4/21 note

On the nature of Numbers

- We all think that we know what a
  - 1, 2, 3, 4 ... are all numbers right!
  - they are representation of numbers
    - you cannot hold a num in hand
- Numbers exist independently of their representation
- Our method for writing numbers derive from Hindu-Arabic numeral system.

Kinds of numbers.

- $\mathbb{N} = \{1, 2, 3, \ldots\}$
- $\mathbb{Z} = \{\ldots -3, -2, -1, 0, 1, 2, 3\}$
- $\mathbb{Q} = \{\frac{a}{b} : a, b \in \mathbb{Z}\}$
- $\mathbb{R} = \mathbb{Q} \cup$ the irrational numbers.
- $\mathbb{C} = \mathbb{R} \cup$ the imaginary numbers

- π, e, γ, Φ are all irrational

Integers and Natural Numbers

- $\mathbb{Z} = \{\ldots -2, -1, 0, 1, 2 \ldots\}$ denotes integers
- $\mathbb{Z}^+ = \{1, 2, 3, \ldots\}$ + integers
- $\mathbb{Z}^- = \{\ldots -3, -2, -1\}$ set of negative integers
- 0 is special it is neither +/-
- $\mathbb{N} = 0 \cup \mathbb{Z}^+$

Computers can represent none of these sets

- Computers do arithmetic in finite field
  - which mean that they are large but finite
- Ex: num with more up to 10 digits
- digit is usually called a bit
  - a bit is either 0 or 1

Positional Number system
every number of N can be written as
$a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b^1 + a_0 b^0$

$1962 = 1\times10^3 + 9\times10^2 + 6\times10^1 + 2\times10^0$
$= 7AA_{16} = 3652_8 = 2200200_3$
$= 11110101010_2$
All represent the same number.

Literal pools
- numbers are written in base 36
- A = 10, B = 11, C = 12, ...
- ADA BC = 10×36^5 + ...

Specifying an integer in C

unsigned } short / long / long long } int

unsigned } char

C Integer data types
char → 8 bits } usually
short → 16 bits
int → at least 16 bits
long → " " 32 "
long long → probably 64 bits

#include <stdint.h>

signed	unsigned	size
int8_t	uint8_t	8 bits
int16_t	uint16_t	16 bits
int32_t		32 bits
		64 bits

Binary arithmetic

Addition works as expected

- 0+0=0
- 1+0=1
- 1+1=10 (add 0 and the 1 is carried)

Multiplication is even simpler:

- 0×0=0
- 1×0=0
- 1×1=1

- 101 + 11 = 1000
- 101 × 101 = 11001

Arithmetic in a Finite field.

- in a finite field, we can define for every number its additive inverse.
  - if j is a number in a finite field and i is its additive inverse then $i + j = 0$.

Suppose we take k bits then we can have every positive integer from 0 to $2^k - 1$

Two's Complement Arithmetic
To get the additive inverse of a number n,
- flip the bits in n, $0 \to 1$, and $1 \to 0$, and then
- add one to result

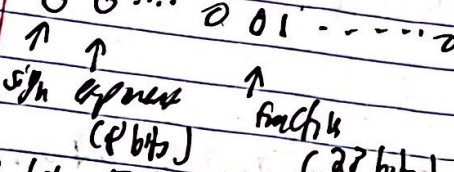
Real Numbers
- Real numbers ($\mathbb{R}$) are:
  - Continuous
  - Uncountably infinite

There are just as many nums between any x and y as there in all of $\mathbb{R}$
- Include all of the
  - Ints ($\mathbb{Z}$)
  - Rational numbers ($\mathbb{Q}$)
  - Irrational nums ($\mathbb{R} - \mathbb{Q}$)

Floating point numbers
- Are a proper subset of the real nums:
  - $\mathbb{FF} \subsetneq \mathbb{R}$
- are a proper subset of rational nums:
  - $\mathbb{FF} \subsetneq \mathbb{Q}$
- are a proper subset of the integers:
  - $\mathbb{FF} \subsetneq \mathbb{Z}$
- mistake to think of a real, rational, they are approximations

Floating point Numbers
float → 32
double → 64
long double → 128

Single precision



Intel Extended Precision (80 bits)
or
Quad (128 bits)

Limits to floating point precision

Big endian, little endian
- Little endian = low address byte
- big endian = high address byte

example

Big endian vs. little endian

Big

12 34 56 78

Little

78 56 34 12

Random number
- True random numbers cannot be created using computer
- Why?
  - Programs are inherently deterministic
  - It has the advantage of repeatability but
  - It has the disadvantage of predictability

Mersenne Twister
- a state of the art pseudo number generator

Arithmetic Operators
- these operators follow the precedence and associativity rules that you learned in high school algebra
- Modulo is the remainder when you divide 2 integers

Type promotion
- when a mix of different types is used, C will promote the lower type to a higher type.

Operator precedence
```
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?:
= += -= *= /= %= ...
```

10/6/21 Notes

Numerical Computation

Basic Arithmetic operation

- Computer can only do basic operation such as add, sub, mult, div
  - mult → shift + add
  - Div → shift + sub
  - add → is little more than exclusive-xor
- trig functions?
  - processor do them internally, thus are still only basic operation

Absolute Value

- abs(x) = if x < 0 then -x else x

Can I just use the library?

- yes, it is best to use the library
- the the library should be designed to be:
  - careful w numerical precision
  - fast + portable

Taylor Series

- if f(x) is a real or complex valued function and it is infinitely differentiable then
  - $f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$
- for example consider e^x centered @ $a = 0$!
  - $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \quad -\infty < x < \infty$

Increasingly better approximation,
- each derivative add a term

e^x

has the simplest taylor series

Computing e^x

- $k! > x^k$ for all x, eventually (when k gets large enough)

so:

- $\frac{x^k}{k!}$ gets smaller w/ each step
- we can stop when it gets small enough that it is "good enough" for our computation

Computing $\sin(x)$

- It is:
  - periodic on $[-2\pi, 2\pi]$, meaning that $\sin(9\pi) = \sin(\pi)$
  - infinitely differentiable, making a good candidate for a Taylor series
- where should

Computing the Terms:

- we skip even numbered terms
- the sign alternates b/t terms
- remove extra extra factors of 2π

Padé approximants

- Padé approximants → ratio of 2 polynomials that correspond to a series but is easier to compute and fit better around the center

What happens if the series converges too slowly?
- Series for $\log(x+1)$:
  - $x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \dots + O(x^n)$
- Converges very slowly unless x is close to 0.
- We can do a little algebra, remember:
  - $\log(x) = -\log\left(\frac{1}{x}\right)$; for $x > 0$
- Recall:
  - $x = \log(e^x) = e^{\log x}$
- You can invert this

Inverting a function
- Simplest function that gives a square:
  - $\sqrt{x}$
- We can't really expand a Taylor series for $x^{1/2}$:

$\sqrt{x}$ w/ binary search
- Guess that it is in the middle.
- Is the guess too small? Look in the larger half.
- Is the guess too big? Look in the smaller half.

$\sqrt{n}$ using a Newton Iterate
- Isaac Newton came to the rescue again.
- We use this iterate:
  - $x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right)$
- As before, we iterate until the approximation is good enough

log(x) using Newton's Method.
- Newton's formula provides a linear approximation of the function
- Solve for x_{n+1}
  - $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- find the root of $f(y) = x - e^y$
- use iterate the equation until answer is close enough.
  - $y_{n+1} = y_n + \frac{x - e^{y_n}}{e^{y_n}}$

Floating point arithmetic
- normal mathematics
  - real numbers are exact
- Computer arithmetic
  - approximation only
- Floating points have round off errors

Round-off Errors
- Errors occur when doing any calculation w/ floating-point
  - rounding errors occur to fit value

*IEEE 754 defines 5 standard rounding modes

Some dangers of floating point arithmetic
add + sub

Comparing floating-point numbers

What do we mean?
- floating point numbers are not real numbers
- we need to take care when working w/ floating point numbers

What should we do?
- use a good library
- understand the different bt floating point and real numbers
  - real = all
  - rational =