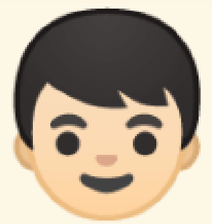
SO..

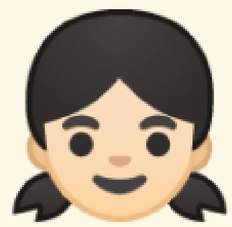
..probably not all of you have kids..

BUT..

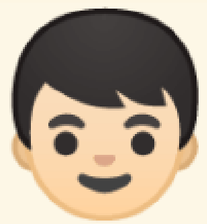
..you might recognize something like this:



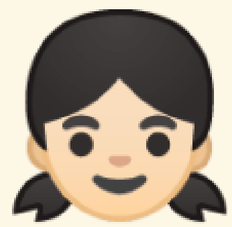
MY SPACESHIP IS FASTER!



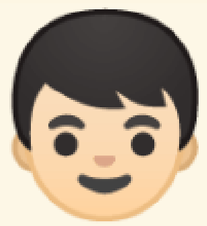
NO! MY SPACESHIP IS FASTER!



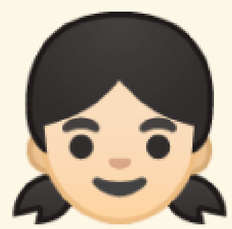
BUT MY SPACESHIP IS FASTER STILL!



AND MY SPACESHIP IS FASTER STILL STILL!



MY SPACESHIP IS ALWAYS ONE HUNDRED FASTER THAN YOURS!



AND MY SPACESHIP IS ALWAYS ONE THOUSAND MILLION FASTER THAN YOURS!

Ad infinitum 😒

HOWEVER!

This is the perfect moment to bring up the topic of recursive functions!

DEFINITION

A recursive function, in logic and mathematics, is a type of function or expression predicating some concept or property of one or more variables, which is specified by a procedure that yields values or instances of that function by repeatedly applying a given relation or routine operation to known values of the function.

Encyclopaedia Britannica

CODING TIME!

RECURSION

Square one

```
long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}


int main() {
  return spaceship_speed(1);
}
```

RECURSION

All the hip kids use lambdas

```
int main() {
  auto spaceship_speed =
    [&](long start_speed) {
      return spaceship_speed(++start_speed);
    };

  return spaceship_speed(1);
}
```

RECURSION

Whoops. C++17 to the rescue!

```
#include <functional>


int main() {
  std::function<long(long)> spaceship_speed =
    [&](long start_speed) {
      return spaceship_speed(++start_speed);
    };

  return spaceship_speed(1);
}
```

RECURSION

Even hipper kids do compile-time code

```
constexpr long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}

int main() {
  return spaceship_speed(1);
}
```

RECURSION

Even hipper kids do compile-time code

```
constexpr long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}

int main() {
  return spaceship_speed(1);
}
```

..or do they?

RECURSION

Uh-oh..

```
constexpr long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}

template<long VALUE>
struct OmgWtf { long value_ = VALUE; };

int main() {
  return OmgWtf<spaceship_speed(1)>{}.value_;
}
```

RECURSION

Oh dear

```
constexpr long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}

template<typename TYPE, TYPE VALUE>
struct OmgWtf { TYPE value_ = VALUE; };

int main() {
  constexpr auto value = spaceship_speed(1);
  return OmgWtf<decltype(value), value>{}.value_;
}
```

RECURSION

Again, all the hip kids use lambdas

```
constexpr long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}

template<typename TYPE, TYPE VALUE>
struct OmgWtf { TYPE value_ = VALUE; };

int main() {
  const auto get_value = []() {
    constexpr auto value = spaceship_speed(1);
    return OmgWtf<decltype(value), value>{}.value_;
  };

  return get_value();
}
```

RECURSION

IIFE to complete the mess

```
constexpr long spaceship_speed(long start_speed) {
  return spaceship_speed(++start_speed);
}

template<typename TYPE, TYPE VALUE>
struct OmgWtf { TYPE value_ = VALUE; };

int main() {
  return [] {
      constexpr auto value = spaceship_speed(1);
      return OmgWtf<decltype(value), value>{}.value_;
    }();
}
```

IDENTIFYING RECRUITMENT OPPORTUNITIES

MULTI-THREADING ISSUES

Multiple children == multithreading issues

TOY-SHARING HELL

Toys are resources; races and deadlocks occur.

TOY-SHARING HELL

Toys are resources; races and deadlocks occur.

All the time.

ANOTHER ONE: RESOURCE OVERCOMMITMENT

Holidays, birthday parties anyone?

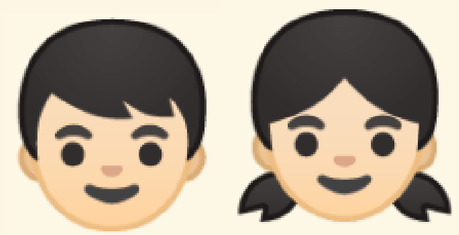
ANOTHER ONE: RESOURCE OVERCOMMITMENT

Holidays, birthday parties anyone?

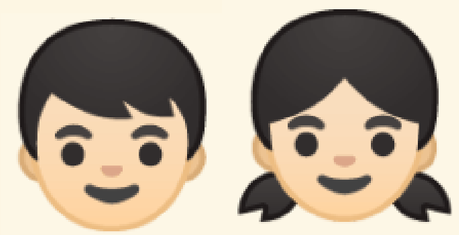
I guess you get it by now.

ONE LAST EXAMPLE

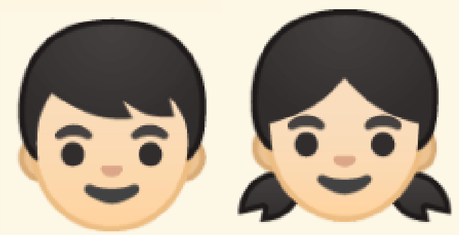
Hey, would you please clean up your toys?



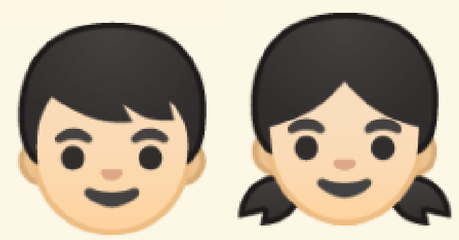
Hey, would you please clean up your toys?



HEY, now will you please clean up your toys?



CLEAN UP YOUR TOYS!



...

Sigh.

signetics

FULLY ENCODED, 9046 X N, RANDOM ACCESS WRITE-ONLY-MEMORY

25120

FINAL SPECIFICATION[10]

DESCRIPTION

The Signetics 25000 Series 9C46XN Random Access Write-Only-Memory employs both enhancement and depletion mode P-Channel, N-Channel and Neu[1] channel MOS devices. Although a static device, a single TTL level clock phase is required to drive the on-board multi-port clock generator. Data refresh is accomplished during CB and LH periods [11]. Quadri-state outputs (when applicable) allow expansion in many directions, depending on organization.

The static memory cells are operated dynamically to yield extremely low power dissipation. All inputs and outputs are directly TL compatible when proper interfacing circuitry is employed.

INPUT PROTECTION

All terminals are provided with slip-on latex protectors for the prevention of Voltage Destruction. (PILL packaged devices do not require protection.)

SILICON PACKAGING

Low cost silicon DIP packaging is implemented and reliability is assured by the use of a non-hermetic sealing technique which prevents the entrapment of harmful ions,, but which allows the free exchange of friendly ions.

SPECIAL FEATURES

Thanks!