

# Why Linear Types Are The Future Of Systems Programming

---

Aditya Siram

February 8, 2021

## Introduction

# Introduction

---

# Standard Hello World

---

```
implement main0 = println! ("hello world")
```

---

# List Datatype

- a linear list type

---

```
datatype list_vt (a:vt@type, int) =  
  | list_nil(a, 0) of ()  
  | {n:int | n >= 0}  
    list_cons(a, n+1) of (a, list(a, n))
```

---

# List Datatype

- probably more familiar

---

```
datatype list_vt =  
  | list_nil      of ()  
  |  
  list_cons      of (a, list(a  ))
```

---

# List Datatype

- indexed on numbers, dependant types!

---

```
datatype list_vt =  
  | list_nil(a, 0) of ()  
  |  
    list_cons(a, n+1) of (a, list(a, n))
```

---

# List Datatype

- numbers can be constrained, refinement types!

---

```
datatype list_vt =  
  | list_nil(a, 0) of ()  
  | {n:int | n >= 0}  
    list_cons(a, n+1) of (a, list(a, n))
```

---



# List Datatype

- parameterized on a view, linear types!

---

```
datatype list_vt_vt (a:vt@type, int) =  
  | list_nil(a, 0) of ()  
  | {n:int | n >= 0}  
    list_cons(a, n+1) of (a, list(a, n))
```

---

# List Datatype

- all ADT's are GADT's in ATS

---

```
datatype list_vt (a:vt@type, int) =  
  | list_nil(a, 0) of ()  
  | {n:int | n >= 0}  
    list_cons(a, n+1) of (a, list(a, n))
```

---

# Manual Memory Management

- What resources are leaked?

---

```
int main(int argc, char** argv) {  
    int* i = (int*)malloc(sizeof(int));  
    *i = 10;  
    FILE* fp = fopen("test.txt", "r");  
    return 0;  
}
```

---

# Manual Memory Management

- Memory!

---

```
int main(int argc, char** argv) {  
    int* i = (int*)malloc(sizeof(int)); // <--- LEAK!!  
    *i = 10;  
    FILE* fp = fopen("test.txt", "r");  
    return 0;  
}
```

---

# Manual Memory Management

- File descriptor

---

```
int main(int argc, char** argv) {  
    int* i = (int*)malloc(sizeof(int)); // <--- LEAK!!  
    *i = 10;  
    FILE* fp = fopen("test.txt", "r"); // <-- LEAK!!  
    return 0;  
}
```

---

# Manual Memory Management

- *Equivalent* ATS program

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (pfset | ()) = ptr_set(pf | i, 10)
  val (pfFile | fp) = fopen("test.txt", "r")
in
  free(pfset | i);
  fclose(pfFile | fp);
end
```

---

# Manual Memory Management

- “Client-facing” code, analogous, no leaks

---

```
implement main0 () = let
  val (      i) = malloc (sizeof<int>)
  val (      ()) = ptr_set(      i, 10)
  val (      fp) = fopen("test.txt", "r")
in
  free(      i);
  fclose(      fp);
end
```

---

# Manual Memory Management

- malloc proof pf passed to ptr\_set

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (      | ()) = ptr_set(pf | i, 10)
  val (      fp) = fopen("test.txt", "r")
in
  free(      i);
  fclose(      fp);
end
```

---



# Manual Memory Management

- malloc proof pf consumed by ptr\_set

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (pfset | ()) = ptr_set(pf | i, 10)
  val (pfFile | fp) = fopen("test.txt", "r")
in
  free(      i);
  fclose(    fp);
end
```

---

# Manual Memory Management

- `fopen` produces a proof of the file descriptor

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (pfset | ()) = ptr_set(pf | i, 10)
  val (pfFile | fp) = fopen("test.txt", "r")
in
  free(pfset | i);
  fclose(pfFile | fp);
end
```

---

# Manual Memory Management

- What happens when `free` and `fopen` are commented out?

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (pfset | ()) = ptr_set(pf | i, 10)
  val (pfFile | fp) = fopen("test.txt", "r")
in
  // free(pfset | i);
  // fclose(pfFile | fp);
end
```

---

# Manual Memory Management

- pfset is left unconsumed

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (pfset <---
  val (pfFile | fp) = fopen("test.txt", "r")
in
  // free(pfset | i);
  // fclose(pfFile | fp);
end
```

---

# Manual Memory Management

- pfFile is left unconsumed

---

```
implement main0 () = let
  val (pf | i) = malloc (sizeof<int>)
  val (pfset <---
  val (pfFile <---
in
  // free(pfset | i);
  // fclose(pfFile | fp);
end
```

---