Cyclone By Example

15-213 Spring 2007

Aleksey Kliger

Computer Science Department
Carnegie Mellon University

March 8, 2007

Outline

From C to Cyclone

Theme

1. Rule out unsafe features of C
2. When possible, check some properties of a program at compile-time.
3. Otherwise, do a runtime check.

Unsafe Features of C

Review of Tuesday's Lecture

1. Pointer arithmetic
2. NULL pointer dereferences
3. Union types
4. Pointers to local variables
5. Casting to pointer types
6. Malloc/free
7. Symbol confusion during linking
8. Format string problems
9. Uninitialized Variables

Unsafe Features of C

Review of Tuesday's Lecture

1. Pointer arithmetic
2. NULL pointer dereferences
3. Union types
4. Pointers to local variables
5. Casting to pointer types
6. Malloc/free
7. Symbol confusion during linking
8. Format string problems
9. Uninitialized Variables

C vs Cyclone

C

1. Pointer arithmetic
2. NULL pointer dereferences
3. Union types
4. Pointers to local variables

Cyclone

1. `@fat` pointers
2. `@notnull` pointers or runtime checks
3. `@tagged` unions
4. `@region()` annotations

Outline

Pointer Arithmetic

C version

Example (Singly-linked lists)

```
struct List {
  int head;
  struct List* tail;
};

typedef
  struct List
  list;
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Pointer Arithmetic

C version

Example (Singly-linked lists)

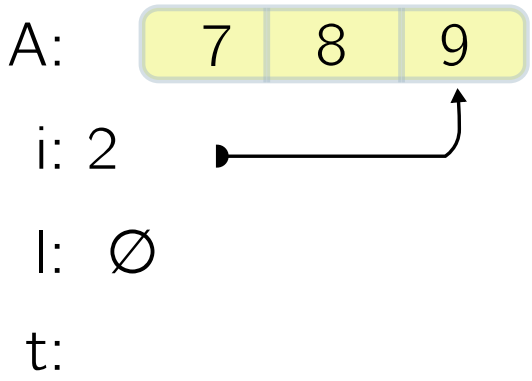
```
struct List {
  int head;
  struct List* tail;
};

typedef
  struct List
  list;
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Illustration



Example

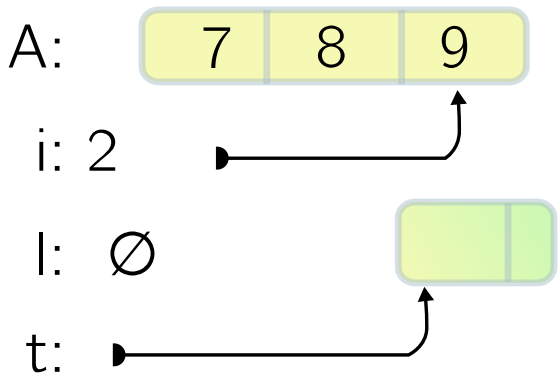
```
int a[3] =
  {7, 8, 9};
list* result =
  array2list (a, 3);
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Illustration



Example

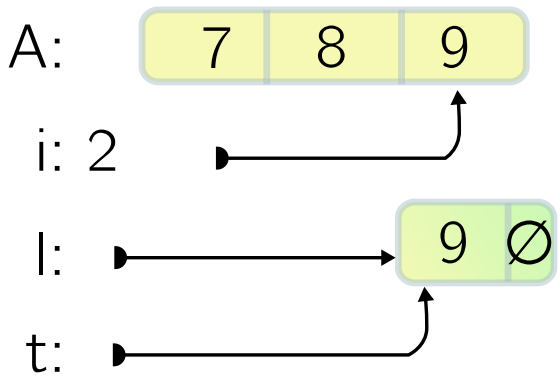
```
int a[3] =
  {7, 8, 9};
list* result =
  array2list (a, 3);
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Illustration



Example

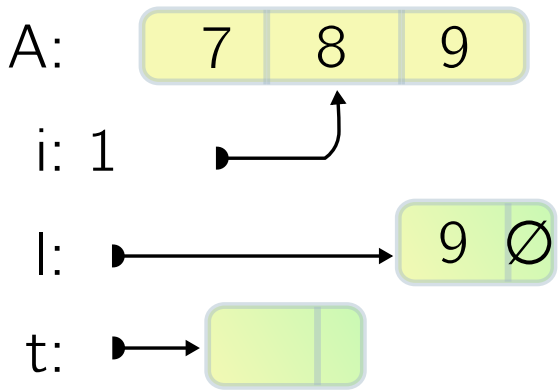
```
int a[3] =
  {7, 8, 9};
list* result =
  array2list (a, 3);
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Illustration



Example

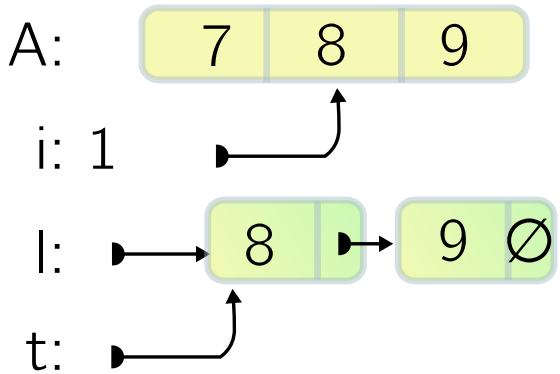
```
int a[3] =
  {7, 8, 9};
list* result =
  array2list (a, 3);
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Illustration



Example

```
int a[3] =
  {7, 8, 9};
list* result =
  array2list (a, 3);
```

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Unsafe Aspects

Question

What's unsafe here?

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Unsafe Aspects

Question

What's unsafe here?

Example (Array to list)

```
list* array2list(int* A, int n) {
  list* l = NULL;
  int i;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Cyclone Version

1. Fat pointer `(int? A)`
2. Number of elements `numelts(A)`
3. Array bounds check

`(int ?` abbreviates `int *@fat)`

Example (Array to list)

```
list* array2list(int? A) {
  list* l = NULL;
  int i;   int n = numelts(A);
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }

  return l;
}
```

Array To List

Let's compile it!

```
[aleksey@tuna] cyclone -Wall -c array2list.cyc
***Warnings***
array2list.cyc:16: inserted bounds check at A[i]
array2list.cyc:16: inserted null check
**************
```

Often, Cyclone can eliminate bounds checks.
Sometimes it can't.

Array To List

Let's compile it!

```
[aleksey@tuna] cyclone -Wall -c array2list.cyc
***Warnings***
array2list.cyc:16: inserted bounds check at A[i]
array2list.cyc:16: inserted null check
**************
```

But why the null check?

Array To List

@fat pointers can be null

If A were NULL, we would never get here, but Cyclone doesn't know that.

Example (Array to list)

```
list* array2list(int? A) {
  list* l = NULL;
  int i;   int n = numelts(A);

  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }
  return l;
}
```

Array To List

@fat pointers can be null

Add our own NULL check (outside the loop).

Example (Array to list)

```
list* array2list(int? A) {
  list* l = NULL;
  int i;   int n = numelts(A);
  if (!A) return NULL;
  for (i = n-1; i >= 0; i--) {
    list* t = malloc (sizeof (list));

    t->head = A[i];
    t->tail = l;
    l = t;
  }
  return l;
}
```

Outline

Zoology

(From a gazelle's perspective)

Example (Animals)

```
struct Herbivore {...};
struct Carnivore {
  int numteeth; ...
};
struct Animal {
  int isCarnivore;
  union {
    struct Herbivore* h;
    struct Carnivore* c;
  };
};
typedef struct Animal
  animal;
```

Example (Count teeth)

```
int count_teeth(animal
                  serengeti[],
                int n) {
  int total = 0;
  int i;

  for (i = 0; i < n; i++) {
    if (serengeti[i].isCarnivore)
      total +=
        serengeti[i].c->numteeth;
  }
  return total;
}
```

Zoology

(From a gazelle's perspective)

Example (Animals)

```
struct Herbivore {...};
struct Carnivore {
  int numteeth; ...
};
struct Animal {
  int isCarnivore;
  union {
    struct Herbivore* h;
    struct Carnivore* c;
  };
};
typedef struct Animal
  animal;
```

Example (Count teeth)

```
int count_teeth(animal
                  serengeti[],
                int n) {
  int total = 0;
  int i;

  for (i = 0; i < n; i++) {
    if (serengeti[i].isCarnivore)
      total +=
        serengeti[i].c->numteeth;
  }
  return total;
}
```

Zoology

Unsafe features

Example (Count teeth)

```
int count_teeth(animal
                  serengeti[],
                int n) {
  int total = 0;
  int i;

  for (i = 0; i < n; i++) {
    if (serengeti[i].isCarnivore)
      total +=
        serengeti[i].c->numteeth;
  }
  return total;
}
```

Zoology

Unsafe features

1. Arrays
2. Tag and union field may be out of sync
3. Dereferencing a pointer without checking for NULL

Example (Count teeth)

```
int count_teeth(animal
                  serengeti[],
                int n) {
  int total = 0;
  int i;

  for (i = 0; i < n; i++) {
    if (serengeti[i].isCarnivore)
      total +=
        serengeti[i].c->numteeth;
  }
  return total;
}
```

Safe Zoology

Use `@tagged` and `@notnull`

Example (Animals)

```
struct Herbivore {...};
struct Carnivore {
  int numteeth; ...
};
struct Animal {
  int isCarnivore;
  union {
    struct Herbivore* h;
    struct Carnivore* c;
  };
};
typedef struct Animal
  animal;
```

Safe Zoology

Use `@tagged` and `@notnull`

Example (Animals)

```
struct Herbivore {...};
struct Carnivore {
  int numteeth; ...
};
@tagged
  union Animal {

    struct Herbivore@ h;
    struct Carnivore@ c;

};
typedef union Animal
  animal;
```

1. `@tagged` union: compiler knows the role of the tag in selecting between `.h` and `.c`
2. `struct Carnivore @` is a pointer that must not be NULL.
   Safe to dereference without a runtime check.

(`struct Carnivore @` is shorthand for `struct Carnivore *@notnull`)

Safe Zoology

Use tagcheck

Example (Animals)

```
struct Herbivore {...};
struct Carnivore {
  int numteeth; ...
};
@tagged
  union Animal {

    struct Herbivore@ h;
    struct Carnivore@ c;

};
typedef union Animal
  animal;
```

Example (Count Teeth)

```
int count_teeth(animal ?
                      serengeti) {
  int total = 0;
  int i;
  int n = numelts(serengeti);
  if (!serengeti) return 0;
  for (i = 0; i < n; i++) {
    if (tagcheck(serengeti[i].c))
      total +=
       serengeti[i].c->numteeth;
  }
  return total;
}
```

Compiling Zoology

Compile

```
[aleksey@tuna] cyclone -Wall -c zoology.cyc
```

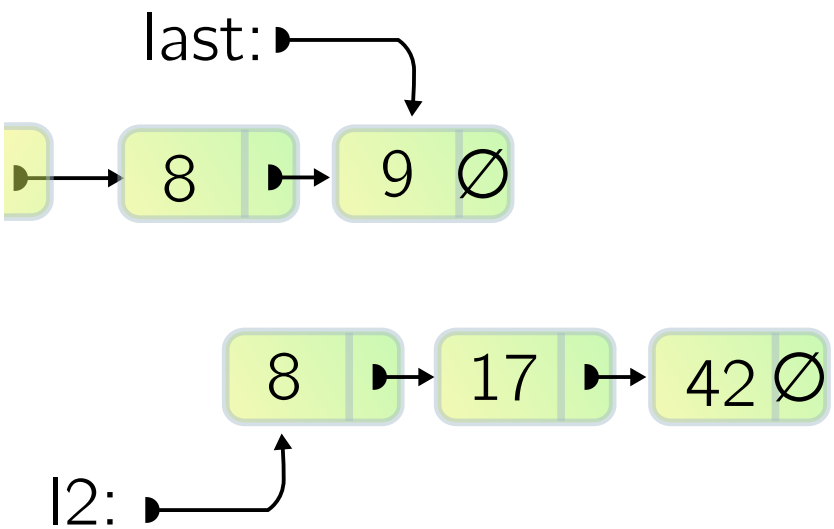
Success!
(Note that Cyclone even figured out that it is
safe to remove the array bounds check)

Outline

Append a list to the end of another

Example

```
void append_to_end (list@ last, list* l2) {
  last->tail = l2;
}
```



That all seems safe...

Compile

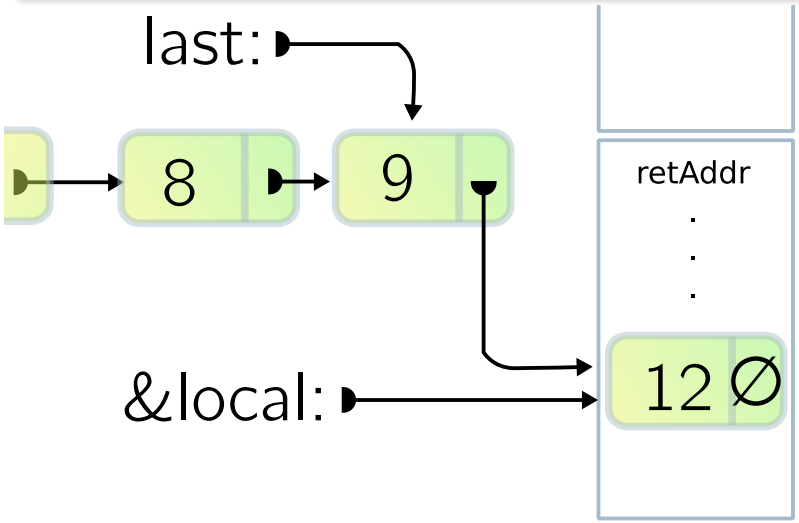
```
[aleksey@tuna]$ cyclone -Wall -c append-to-end.cyc

append-to-end.cyc:12: type mismatch:
  struct List * != list *‘GE1
  ‘GE1 and ‘H are not compatible.

COMPILATION FAILED!
```

What in the world is ‘GE1?

It's a pointer to where?

Example

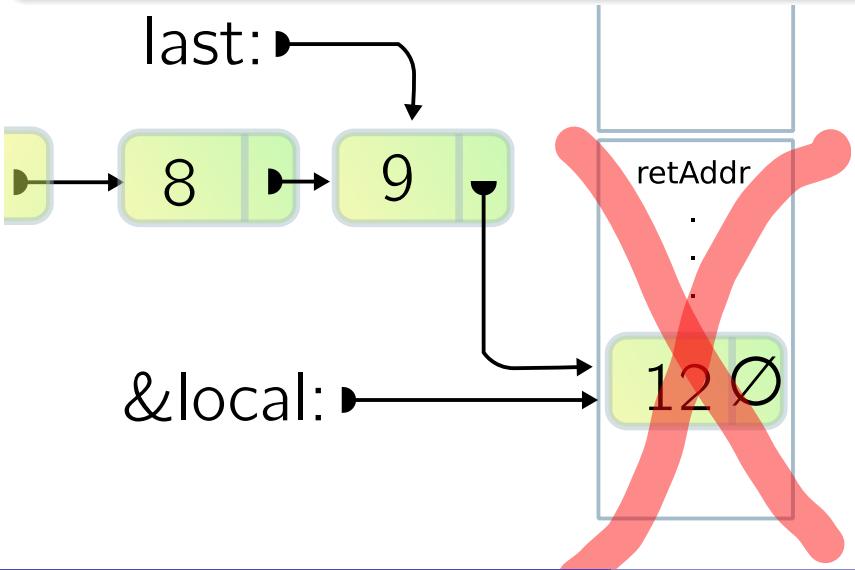
```
...
list local;
local.head = 12;
local.tail = NULL;
append_to_end(last, &local);
...
return ...
```



It's a pointer to where?

Example

```
...
list local;
local.head = 12;
local.tail = NULL;
append_to_end(last, &local);
...
return ...
```



Regions

Every pointer has a region:

`@region(‘H)` pointer to the heap

`@region(‘func_name)` pointer to the stack frame of `func_name()`

`@regions(‘regionvar)` pointer to some arbitrary region

By default:

- Function arguments point to an arbitrary region
- Function returns point to the heap
- Pointers mentioned in `structs`, `unions`, and `typedefs` point to the heap

Error Message Revisited

Example

```
void append_to_end (list@ last, list*‘reg_var l2) {
  last->tail = l2;
}
```

Compile

```
[aleksey@tuna]$ cyclone -Wall -c append-to-end.cyc

append-to-end.cyc:12: type mismatch:
  struct List * != list *‘reg_var
  ‘reg_var and ‘H are not compatible.

COMPILATION FAILED!
```

Typical solution to region errors

Put it on the heap

Example

```
void append_to_end (list@ last, list*`H l2) {
  last->tail = l2;
}
```

Compile

```
[aleksey@tuna]$ cyclone -Wall -c append-to-end.cyc
```

Success!

Summary

Pointer Arithmetic Fat pointers and bounds checks

Unions of pointers Tagged unions

Dereferencing NULL Not-NULL pointers and runitme checks

Returning pointers to local variables Region annotations

Cyclone defaults sometimes too generous: lead to errors.

Add `'H`

Questions?

The Cyclone homepage is `http://cyclone.thelanguage.org`