The Tiger Example Front-end for C--

Simple language from Andrew Appel.

Sample compiler with lots of documentation.

Emphasize features of C--

Provide a reference for developers.

Problems solved using C- - features

Find roots for garbage collection

- runtime interface

Communicate pointer information to GC

- runtime interface
- user spans

Implement Exceptions

- runtime interface
- control flow (cut to, unwind)

Inter-operate with C code

- multiple calling conventions

Control stack layout for nested functions

- stackdata

What is Tiger

- Expression-based language
- Nested functions (PASCAL)
- Monomorphic type system
- Dynamic allocation of records, arrays
- Garbage collected
- Exceptions

```
function add_mod_256(x:int,y:int):int =
  let var z := x + y in
  if z >= 256 then z - 256 else z end
```

Types in Tiger

integers and strings

user-defined arrays:

```
type int_array = array of int
var int_array[10] of 0
```

user-defined records:

```
type person = { name : string, age : int }
var js := { name = "John Smith", age = 35 }
```

Arrays, records allocated on the heap

Memory reclaimed by simple copying collector

Representation in Memory

All types 32-bit representation: pointer,integer

Everything can be placed in C- - register variable

All heap objects multiple of 32-bits

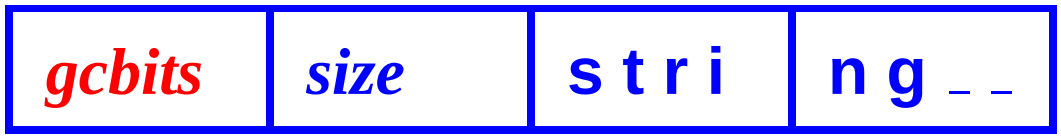
***No* chars, floats, ...**

Representation in Memory

integers :

C- - variables

strings:

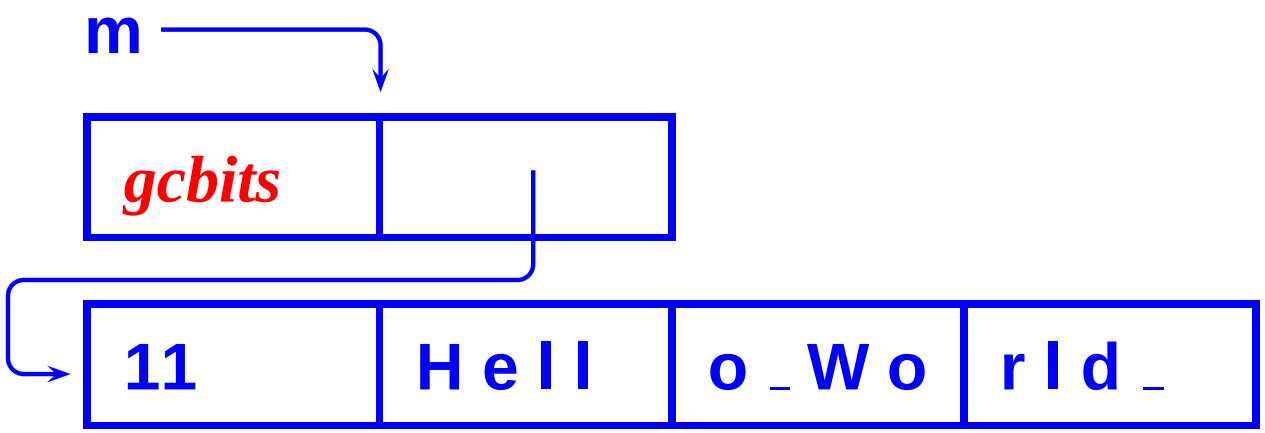


arrays:

records:

Allocation in Tiger

```
let
  type msg = { str : string }
  var m := msg { str = "Hello World" }
in
  print(m.str)
end
```



Allocation in C--

```
bits32 alloc_ptr; /* free space pointer */
tiger_main() {
  bits32 rv;
  bits32 m;
  if (alloc_ptr + 8 > bits32[space_end]) {
    call_gc();
  }
  bits32[alloc_ptr] = 4; /* length in bytes */
  bits32[alloc_ptr + 4] = Lgbl_21;
  m = alloc_ptr + 4;
  alloc_ptr = alloc_ptr + 8;
  rv = foreign"C" print(bits32[m], 0);
  return(rv);
}
```

Allocation in C--

```
bits32 alloc_ptr; /* free space pointer */
tiger_main() {
  bits32 rv;
  bits32 m;
  if (alloc_ptr + 8 > bits32[space_end]) {
    call_gc();
  }
  bits32[alloc_ptr] = 4; /* length in bytes */
  bits32[alloc_ptr + 4] = Lgbl_21;
  m = alloc_ptr + 4;
  alloc_ptr = alloc_ptr + 8;
  rv = foreign"C" print(bits32[m], 0);
  return(rv);
}
```

Allocation in C--

```
bits32 alloc_ptr; /* free space pointer */
tiger_main() {
  bits32 rv;
  bits32 m;
  if (alloc_ptr + 8 > bits32[space_end]) {
    call_gc();
  }
  bits32[alloc_ptr] = 4; /* length in bytes */
  bits32[alloc_ptr + 4] = Lgbl_21;
  m = alloc_ptr + 4;
  alloc_ptr = alloc_ptr + 8;
  rv = foreign"C" print(bits32[m], 0);
  return(rv);
}
```

Allocation in C--

```
bits32 alloc_ptr; /* free space pointer */
tiger_main() {
  bits32 rv;
  bits32 m;
  if (alloc_ptr + 8 > bits32[space_end]) {
    call_gc();
  }
  bits32[alloc_ptr] = 4; /* length in bytes */
  bits32[alloc_ptr + 4] = Lgbl_21;
  m = alloc_ptr + 4;
  alloc_ptr = alloc_ptr + 8;
  rv = foreign"C" print(bits32[m], 0);
  return(rv);
}
```

Allocation in C--

```
bits32 alloc_ptr; /* free space pointer */
tiger_main() {
  bits32 rv;
  bits32 m;
  if (alloc_ptr + 8 > bits32[space_end]) {
    call_gc();
  }
  bits32[alloc_ptr] = 4; /* length in bytes */
  bits32[alloc_ptr + 4] = Lgbl_21;
  m = alloc_ptr + 4;
  alloc_ptr = alloc_ptr + 8;
  rv = foreign"C" print(bits32[m], 0);
  return(rv);
}
```

Allocation C- -

```
call_gc() {
  alloc_ptr = foreign "C" tig_gc(k) also cuts to k;
  return;
continuation k():
  return;
}
```

Read stack starting at k

Saves callee-saves registers

Must save and restore globals

Garbage Collection

Tiger GC gets possible roots from C- - runtime.

- **Function Parameters**
- **Local variables**
- **Stack labels**
- **Globals**

Only front end knows which variables are pointers

Pointer information in user span data

User Span Data

Initialized data associated with spans of C-- code

```
const SPAN_ID = 1
section "data" { span_data : ...; }

cmm_code() {
 span SPAN_ID span_data {
    ...
 }
}
```

Accessing Span Data

Span data access through C-- runtime interface

```
#define SPAN_ID 1

void runtime_system(Cmm_Cont* k) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  unsigned* span_data =
          Cmm_GetDescriptor(&a, SPAN_ID);
  ...
}
```

Tiger GC Data

```
tiger_main() {
 span GC tiger_main_gc_data {
   bits32 rv;
   bits32 m;
   ...
 }
}
section "data" {
 tiger_main_gc_data: bits32[] { 0,1 };
}
```

Implementing Garbage Collection

Call GC from allocator

```
call_gc() {
  alloc_ptr = foreign "C" tig_gc(k) also cuts to k;
  return;
continuation k():
  return;
}
```

Use C-- runtime interface to walk stack

Pointer information from user spans

Perform normal GC cycle

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

GC Main Loop

```
void* tig_gc(Cmm_Cont* k) {
  c_alloc_ptr = to_space;
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
   unsigned var_count = Cmm_LocalVarCount(&a);
   unsigned* gc_data = Cmm_GetDescriptor(&a, GC);
   for (i = 0; i < var_count; ++i) {
     unsigned** rootp = Cmm_FindLocalVar(&a, i);
     *rootp = maybe_gc_forward(*rootp, gc_data[i]);
     Cmm_LocalVarWritten(&a, i);
   }
  } while(Cmm_ChangeActivation(&a));
  gc_copy(); gc_flip();
  return c_alloc_ptr;
}
```

Starting up Tiger

C- - flexible calling conventions

=

Lots of choices

Call from C language main

Specify convention for calling from Your Favorite Language

Write startup code in C- -

Tiger Startup Code

```
foreign "C" main(bits32 argc, bits32 argv) {
  bits32 ret_val;
  foreign "C" gc_init(8192);
  tig_set_handler(unhandled);
  ret_val = tiger_main(0) also cuts to unhandled;
  foreign "C" return (ret_val);
continuation unhandled(ret_val):
  foreign "C" printf(exn_msg, ret_val);
  foreign "C" return(-1);
}
```

Tiger Startup Code

```
foreign "C" main(bits32 argc, bits32 argv) {
  bits32 ret_val;
  foreign "C" gc_init(8192);
  tig_set_handler(unhandled);
  ret_val = tiger_main(0) also cuts to unhandled;
  foreign "C" return (ret_val);
continuation unhandled(ret_val):
  foreign "C" printf(exn_msg, ret_val);
  foreign "C" return(-1);
}
```

Tiger Startup Code

```
foreign "C" main(bits32 argc, bits32 argv) {
  bits32 ret_val;
  foreign "C" gc_init(8192);
  tig_set_handler(unhandled);
  ret_val = tiger_main(0) also cuts to unhandled;
  foreign "C" return (ret_val);
continuation unhandled(ret_val):
  foreign "C" printf(exn_msg, ret_val);
  foreign "C" return(-1);
}
```

Tiger Startup Code

```
foreign "C" main(bits32 argc, bits32 argv) {
  bits32 ret_val;
  foreign "C" gc_init(8192);
  tig_set_handler(unhandled);
  ret_val = tiger_main(0) also cuts to unhandled;
  foreign "C" return (ret_val);
continuation unhandled(ret_val):
  foreign "C" printf(exn_msg, ret_val);
  foreign "C" return(-1);
}
```

Tiger Startup Code

```
foreign "C" main(bits32 argc, bits32 argv) {
  bits32 ret_val;
  foreign "C" gc_init(8192);
  tig_set_handler(unhandled);
  ret_val = tiger_main(0) also cuts to unhandled;
  foreign "C" return (ret_val);
continuation unhandled(ret_val):
  foreign "C" printf(exn_msg, ret_val);
  foreign "C" return(-1);
}
```

Exceptions: the implementation space

Execute in	Stack walk required? No	Stack walk required? Yes
Generated code	`cut to`	`return <`*m*`/`*n*`>`
Run-time system	`CutTo`	`MakeUnwindCont`

Exceptions in Tiger

Two implementations cut to, unwind

cut to

- **small runtime cost to enter try block**
- **raising exception very cheap**

unwind

- **no runtime cost to enter try block**
- **high cost to raise exception**

Switching is *easy* with C- -

An Example of Exceptions

```
let
  exception ex1
  exception ex2
  function f(i : int) =
    if (i > 0) then raise ex1 else raise ex2
in
 try f(1)
 handle ex1 print("exception 1\n") end
 handle ex2 print("exception 2\n") end;
end
```

Exceptions (cut to)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;
  bits32 old_handler;
  old_handler = tig_set_handler(handler);
  f(1) also cuts to handler;
  tig_set_handler(old_handler);
  goto Ltry_end;

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (cut to)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;
  bits32 old_handler;
  old_handler = tig_set_handler(handler);
  f(1) also cuts to handler;
  tig_set_handler(old_handler);
  goto Ltry_end;

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (cut to)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;
  bits32 old_handler;
  old_handler = tig_set_handler(handler);
  f(1) also cuts to handler;
  tig_set_handler(old_handler);
  goto Ltry_end;

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (cut to)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;
  bits32 old_handler;
  old_handler = tig_set_handler(handler);
  f(1) also cuts to handler;
  tig_set_handler(old_handler);
  goto Ltry_end;

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (cut to)

```
continuation handler(exn_id):
  tig_set_handler(old_handler);
  if (exn_id == EXN_ex1) {
    foreign "C" print(Lgbl_36) ;
    goto Ltry_end;
  }
  if (exn_id == EXN_ex2) {
    foreign "C" print(Lgbl_37);
    goto Ltry_end;
  }
  tig_raise(exn_id); /* default handler */
  goto Ltry_end;
```

Exceptions (cut to)

```
continuation handler(exn_id):
  tig_set_handler(old_handler);
  if (exn_id == EXN_ex1) {
    foreign "C" print(Lgbl_36) ;
    goto Ltry_end;
  }
  if (exn_id == EXN_ex2) {
    foreign "C" print(Lgbl_37);
    goto Ltry_end;
  }
  tig_raise(exn_id); /* default handler */
  goto Ltry_end;
```

Exceptions (cut to)

```
continuation handler(exn_id):
  tig_set_handler(old_handler);
  if (exn_id == EXN_ex1) {
    foreign "C" print(Lgbl_36) ;
    goto Ltry_end;
  }
  if (exn_id == EXN_ex2) {
    foreign "C" print(Lgbl_37);
    goto Ltry_end;
  }
  tig_raise(exn_id); /* default handler */
  goto Ltry_end;
```

Exceptions (cut to)

```
continuation handler(exn_id):
  tig_set_handler(old_handler);
  if (exn_id == EXN_ex1) {
    foreign "C" print(Lgbl_36) ;
    goto Ltry_end;
  }
  if (exn_id == EXN_ex2) {
    foreign "C" print(Lgbl_37);
    goto Ltry_end;
  }
  tig_raise(exn_id); /* default handler */
  goto Ltry_end;
```

Exceptions (cut to)

```
continuation handler(exn_id):
  tig_set_handler(old_handler);
  if (exn_id == EXN_ex1) {
    foreign "C" print(Lgbl_36) ;
    goto Ltry_end;
  }
  if (exn_id == EXN_ex2) {
    foreign "C" print(Lgbl_37);
    goto Ltry_end;
  }
  tig_raise(exn_id); /* default handler */
  goto Ltry_end;
```

Exceptions (cut to)

```
section "data" { curr_exn : bits32; }
tig_set_handler(bits32 exn) {
  bits32 old_exn;
  old_exn = bits32[curr_exn];
  bits32[curr_exn] = exn;
  return(old_exn);
}

tig_raise(bits32 exn_id) {
  cut to bits32[curr_exn](exn_id);
}
```

Exceptions (cut to)

```
section "data" { curr_exn : bits32; }
tig_set_handler(bits32 exn) {
  bits32 old_exn;
  old_exn = bits32[curr_exn];
  bits32[curr_exn] = exn;
  return(old_exn);
}

tig_raise(bits32 exn_id) {
  cut to bits32[curr_exn](exn_id);
}
```

Exceptions (cut to)

```
section "data" { curr_exn : bits32; }
tig_set_handler(bits32 exn) {
  bits32 old_exn;
  old_exn = bits32[curr_exn];
  bits32[curr_exn] = exn;
  return(old_exn);
}

tig_raise(bits32 exn_id) {
  cut to bits32[curr_exn](exn_id);
}
```

Exceptions (cut to)

```
section "data" { curr_exn : bits32; }
tig_set_handler(bits32 exn) {
  bits32 old_exn;
  old_exn = bits32[curr_exn];
  bits32[curr_exn] = exn;
  return(old_exn);
}

tig_raise(bits32 exn_id) {
  cut to bits32[curr_exn](exn_id);
}
```

Exceptions in Tiger

Two implementations cut to, unwind

cut to

- **small runtime cost to enter try block**
- **raising exception very cheap**

unwind

- **no runtime cost to enter try block**
- **high cost to raise exception**

Switching is *easy* with C- -

Exceptions (unwind)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;

span EXN exn_data {
  f(1) also unwinds to handler;
  goto Ltry_end;
}

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (unwind)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;

span EXN exn_data {
  f(1) also unwinds to handler;
  goto Ltry_end;
}

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (unwind)

```
tiger_main(bits32 pfp) {
  bits32 exn_id;

span EXN exn_data {
  f(1) also unwinds to handler;
  goto Ltry_end;
}

continuation handler(exn_id):
  ...
  goto Ltry_end;

Ltry_end:
  return(0);
}
```

Exceptions (unwind)

```
void tig_raise(Cmm_Cont* k, unsigned exn_id) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
    unsigned* exn_data = Cmm_GetDescriptor(&a, EXN);
    if (exn_in_activation(exn_id, exn_data)) {
      unsigned n = activation_number(exn_id, exn_data);
      Cmm_Cont* exn = MakeUnwindCont(a, n, exn_id);
      Cmm_CutTo(exn);
      return;
    }
  } while(Cmm_ChangeActivation(&a));
  return NULL;
}
```

Exceptions (unwind)

```
void tig_raise(Cmm_Cont* k, unsigned exn_id) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
    unsigned* exn_data = Cmm_GetDescriptor(&a, EXN);
    if (exn_in_activation(exn_id, exn_data)) {
      unsigned n = activation_number(exn_id, exn_data);
      Cmm_Cont* exn = MakeUnwindCont(a, n, exn_id);
      Cmm_CutTo(exn);
      return;
    }
  } while(Cmm_ChangeActivation(&a));
  return NULL;
}
```

Exceptions (unwind)

```
void tig_raise(Cmm_Cont* k, unsigned exn_id) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
    unsigned* exn_data = Cmm_GetDescriptor(&a, EXN);
    if (exn_in_activation(exn_id, exn_data)) {
      unsigned n = activation_number(exn_id, exn_data);
      Cmm_Cont* exn = MakeUnwindCont(a, n, exn_id);
      Cmm_CutTo(exn);
      return;
    }
  } while(Cmm_ChangeActivation(&a));
  return NULL;
}
```

Exceptions (unwind)

```
void tig_raise(Cmm_Cont* k, unsigned exn_id) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
    unsigned* exn_data = Cmm_GetDescriptor(&a, EXN);
    if (exn_in_activation(exn_id, exn_data)) {
      unsigned n = activation_number(exn_id, exn_data);
      Cmm_Cont* exn = MakeUnwindCont(a, n, exn_id);
      Cmm_CutTo(exn);
      return;
    }
  } while(Cmm_ChangeActivation(&a));
  return NULL;
}
```

Exceptions (unwind)

```
void tig_raise(Cmm_Cont* k, unsigned exn_id) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
    unsigned* exn_data = Cmm_GetDescriptor(&a, EXN);
    if (exn_in_activation(exn_id, exn_data)) {
      unsigned n = activation_number(exn_id, exn_data);
      Cmm_Cont* exn = MakeUnwindCont(a, n, exn_id);
      Cmm_CutTo(exn);
      return;
    }
  } while(Cmm_ChangeActivation(&a));
  return NULL;
}
```

Exceptions (unwind)

```
void tig_raise(Cmm_Cont* k, unsigned exn_id) {
  Cmm_Activation a = Cmm_YoungestActivation(k);
  do {
    unsigned* exn_data = Cmm_GetDescriptor(&a, EXN);
    if (exn_in_activation(exn_id, exn_data)) {
      unsigned n = activation_number(exn_id, exn_data);
      Cmm_Cont* exn = MakeUnwindCont(a, n, exn_id);
      Cmm_CutTo(exn);
      return;
    }
  } while(Cmm_ChangeActivation(&a));
  return NULL;
}
```

Exception Implementations Comparison

	cut to	unwind
enter block	push and pop	none
raise	stack cut	walk stack

Can easily switch between implementations

```
qc-- tiger.lua foo.tig
qc-- tiger.lua Tiger.unwind=1 foo.tig
```

Conclusion

Section	Lines of Code
Front end	3213
Code Generator	241
Runtime (generic)	223
Runtime (C- - specific)	178

11% of code specific to C- -

Other Tiger Compilers:

João Dias' 30% for generating Alpha Code

Matthew Fluet's 50% for generating Mips Code