CSCI 2041: Exception Handling

Chris Kauffman

Last Updated:
Fri Nov 30 10:26:08 CST 2018
Logistics

Reading
Practical OCaml: Ch 10
Exception Handling

Goals
- Exception Handling
- Control Flow

A5: Calculon
- Arithmetic language interpreter
- 2X credit for assignment
- 5 Required Problems 100pts
- 5 Option Problems 50pts
- Milestone deadline Wed 12/5
- Final deadline Tue 12/11
Exceptions

- Exceptions are a means to alter control flow usually associated with errors.
- Widely used by most modern programming environments.
- Briefly discuss raising or "throwing" exceptions and defining our own kinds.
- Most often you will need to handle or "catch" exceptions raised by library code so will spend significant time on this.
Declaration and Raising

- Declare with the keyword `exception` and endow with data via same syntax as algebraic types.
- Raise exceptions with `raise` keyword

```ocaml
(* declaration is similar to syntax for algebraic types *)
exception Screwup;;

exception Meltdown of int;;

exception Kersplosion of { radius : float;
  damage : int; };;

(* keyword raise will raise an exception *)
raise Screwup;;

raise (Meltdown 5);;

raise (Meltdown 2);;

raise (Kersplosion{radius=5.46; damage=20});;

let e = Meltdown 2 in (* create, don’t raise *)
raise e;; (* now raise *)
```
Two Alternatives

- Recall the assoc operation: look up a value based on a key in a list of pairs like this one
  
  ```ocaml
  let alist = ["Bulbasaur", "Grass");
  ("Squirtle", "Water");
  ("Charmander", "Fire");]
  ;;
  ```

- Contrast List module’s assoc_opt and assoc below in functions from print_kind.ml

- Note that assoc may raise a Not_found exception which should be handled in a try/with block

- Experiment with these two in the REPL

```ocaml
1 (* look up kind using assoc_opt *)  1 (* look up kind using assoc; catch exceptions *)
2 let print_kind1 pok =                2     try
3   printf "%s: " pok;               3     let kind = assoc pok alist in
4   let result = assoc_opt pok alist in 4     printf "%s\n" kind;
5   match result with                 5     with
6     | None -> printf "Unknown\n"       6     | Not_found -> printf "Unknown\n"
7     | Some(kind) -> printf "%s\n" kind  7     | Not_found -> printf "Unknown\n"
8   ;;                                 8     with
9
10```

```ocaml
1 (* look up kind using assoc; catch exceptions *)
2 let print_kind2 pok =                2     try
3   printf "%s: " pok;               3     let kind = assoc pok alist in
4   try                               4     printf "%s\n" kind;
5     let kind = assoc pok alist in    5     with
6     try                               6     | Not_found -> printf "Unknown\n"
7     let kind = assoc pok alist in    7     | Not_found -> printf "Unknown\n"
8     printf "%s\n" kind               8     with
9     | Not_found -> printf "Unknown\n"   9     | Not_found -> printf "Unknown\n"
10
11```
Both functions work identically

Print "Unknown" when there is something missing from the list
Error-Checking

- assoc_opt follows the old-school approach
  - run a function
  - check immediately whether it succeeded
  - handle errors if things went sideways
- This is how non-exception languages like C deal with errors

```c
while(1){
  printf("query: ");
  result = fscanf(stdin,"%s",buf);
  if(result==EOF){ // check for error
    printf("end of file\n");
    break;
  }

  char *input = malloc(strlen(buf));
  if(input == NULL){ // check for error
    printf("out of memory");
    exit(1);
  }
}
```

1. Error-checking is error-prone and tedious
2. No separation of error generation from error policy: e.g. have to handle errors immediately where they are identified without broader context: motivates exceptions
A Third, Tempting Alternative

- OCaml does not require exception-generating code to be wrapped in a try/with block\(^1\)
- This allows below version `print_kind3` to use assoc sans try/with
- In a REPL, this appears to have no major effect other than not printing `Unknown` like the previous versions

```ocaml
(* look up kind but don’t catch exceptions *)
let print_kind3 pok =
  printf "%s: " pok;
  let kind = assoc pok alist in
  printf "%s\n" kind;
;;

# print_kind3 "Squirtle";;
Squirtle: Water
- : unit = ()

# print_kind3 "Charmander";;
Charmander: Fire
- : unit = ()

# print_kind3 "Jigglypuff";;
Jigglypuff: Exception: Not_found.
#
```

\(^1\)OCaml uses *unchecked exceptions* like most programming languages aside from Java which also has *checked exceptions* which must be either caught or declared in function prototypes.
Consequences of not Catching

- Despite the innocuous appearance in the REPL, exceptions can have dire consequences in programs.
- An **unhandled / uncaught exception** typically ends a program (to the dismay of users).

```ocaml
(* main loop which asks for repeated input *)
let _ =
  let quit_now = ref false in
  while not !quit_now do
    printf "query: ";
    let pok = read_line () in
    if pok="quit" then
      quit_now := true
    else
      begin
        print_kind1 pok;
        print_kind2 pok;
        print_kind3 pok (* !! *)
      end;
  done;
;;
```

> `ocamlc print_kind.ml print_kind_main.ml

> a.out

query: Charmander
Charmander: Fire
Charmander: Fire
Charmander: Fire

query: Bulbasaur
Bulbasaur: Grass
Bulbasaur: Grass
Bulbasaur: Grass

query: Pikachu
Pikachu: Unknown
Pikachu: Unknown
Pikachu: Fatal error: exception Not_found

>
Getting Exception Backtraces in OCaml

- A **backtrace** shows what functions were active when an exception was thrown.
- Useful when programs crash to help diagnose the path to the error condition.
- OCaml **disables backtraces** by default.
  - Performance is improved by this decision.
  - Most other languages w/ exceptions enable backtraces by default to assist with debugging.
- Compile with debugging information: `ocamlc -g`.
- Enable backtrace printing in one of two ways:
  1. Via environment variable `OCAMLRUNPARAM`:
     - `ocamlc -g prog.ml`
     - `export OCAMLRUNPARAM=b`
     - `./a.out`
  2. In source code, call `record_backtrace`:
     ```ocaml```
     ```
     Printexc.record_backtrace true;;
     ```
- Exceptions that cause the program to crash produce a listing of the functions that were active at the time of the crash.
Example: Backtrace for print_kind_main.ml

Not going to edit the source code so enable backtraces via command line

> ocamlc -g print_kind.ml print_kind_main.ml # compile with debug info
> export OCAMLRUNPARAM=b # set env var to enable backtraces
> a.out # run program
query: Squirtle
Squirtle: Water
Squirtle: Water
Squirtle: Water
query: Jigglypuff # not found
Jigglypuff: Unknown
Jigglypuff: Unknown
Jigglypuff: Fatal error: exception Not_found # BACKTRACE
Raised at file "list.ml", line 187, characters 16-25 # origin
Called from file "print_kind.ml", line 35, characters 13-28 # active func
Called from file "print_kind_main.ml", line 17, characters 8-23 # active func
Exercise: Exceptions Percolate Up

- Exceptions work their way up the call stack
- On the way up, applicable try/with blocks are consulted to see if they can handle the exception
- Note that the raise location may be very different from the handle position and may be many function calls away
- What else can go wrong in the main loop?

```ocaml
let _ = (* inner_catch.ml *)
let quit_now = ref false in
while not !quit_now do
    printf "query: ";
    let pok = read_line () in
    if pok="quit" then
        quit_now := true
    else
        try (* begin try *)
            print_kind3 pok (* may throw *)
        with (* exc handling *)
        | Not_found -> printf "Oops!\n";
    done;
;;
```

```
> ocamlc print_kind.ml \ inner_catch.ml
> a.out
query: Bulbasaur
Bulbasaur: Grass
query: Jigglypuff
Jigglypuff: Oops!
query: Pikachu
Pikachu: Oops!
query: Mewtwo
Mewtwo: Oops!
query: quit
```
Pressing Ctrl-d sends "End of file" character to indicate no more input. Causes read_line to raise an exception

How can this be "fixed"?
Answers: Several Things May Go Wrong

- print_kind3 may raise Not_found
- read_line may raise End_of_file
- May want to catch both of them

```ocaml
let _ = (* separate_catch.ml *)
let quit_now = ref false in
while not !quit_now do
  printf "query: ";
  let pok =
    try (* begin try *)
      read_line () (* may throw *)
    with (* exc handling *)
      | End_of_file -> "Default"
    in
    if pok="quit" then
      quit_now := true
    else
      try (* begin try *)
        print_kind3 pok (* may throw *)
      with (* exc handling *)
        | Not_found -> printf "Oops!\n"
    done;
```

- Starts getting ugly style-wise, like the C-style of immediate error handling after running a function
There are Many Kinds of Exceptions

Exception types are like algebraic variants

- Can carry data, match individual types in try/with
- **No warnings** for missing a relevant type of exception

```ocaml
let _ =
let quit_now = ref false in
while not !quit_now do
  printf "query: ";
  let pok = read_line () in
  if pok="quit" then
    quit_now := true
  else
    try
      print_kind3 pok
    with (* no handlers apply to Not_found *)
    | Failure msg -> printf "Error: %s!\n" msg;
    | Invalid_argument a -> printf "Invalid arg!\n";
    done;

---DEMO---
> ocamlc print_kind.ml wrong_exc.ml
> a.out
query: Pikachu
Pikachu: Fatal error: exception Not_found
```
Handle/Catch Cases are like `match/with`

- Match exception specific kinds to appropriate actions
- May include a "catch-all" case with continue or exit actions

```ocaml
let _ = (* catch_em_all.ml *)
let quit_now = ref false in
while not !quit_now do
  try (* begin try *)
    printf "query: ";
    let pok = read_line () in (* may throw End_of_file *)
    if pok="quit" then
      quit_now := true
    else
      print_kind3 pok; (* may throw Not_found *)
    with (* exc handling *)
    | Not_found -> printf "Oops!\n";
    | End_of_file -> printf "Catch more!\n"
    | exc -> (* catch any other exception *)
      printf "\nSomething went wrong somewhere!\n";
      let excstr = Printexc.to_string exc in
      printf "\nException: %s\n" excstr;
  (* keep looping after reporting exception *)
  done;
```
Error generation

- Calling read_line returns a string but may raise End_of_file
- Calling print_kind3 may raise a Not_found

Error-handling policy **specific to this program**

- Print "Oops" for Not_found
- Print "Don’t leave.." for End_of_file

Other programs can establish different error-handling policies like Quit on End_of_file

```
> ocamlc print_kind.ml catch_em_all.ml
> a.out
query: Bulbasaur
Bulbasaur: Grass
query: Pikachu
Pikachu: Oops!
query: Catch more!
query: Catch more!
query: Squirtle
Squirtle: Water
query: quit
```

Exceptions separate error generation and error handling allowing program-specific policies to handle the same kinds of errors