Control in Sequential Languages
Exceptions

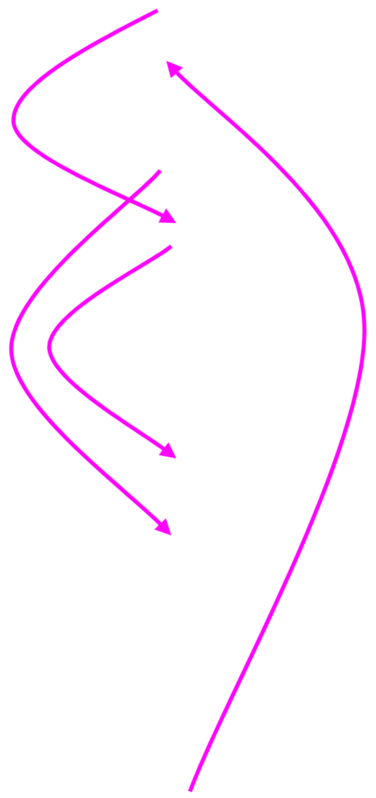
Angelo Gargantini

Topics cap 8

- Structured Programming
  - Go to considered harmful
- Exceptions
  - "structured" jumps that may return a value
  - dynamic scoping of exception handler
- ~~Continuations~~
  - ~~Function representing the rest of the program~~
  - ~~Generalized form of tail recursion~~
- ~~Control of evaluation order (force and delay)~~
  - ~~May not cover in lecture. Book section straightforward.~~

Fortran Control Structure

```
10 IF (X .GT. 0.000001) GO TO 20
11 X = -X
   IF (X .LT. 0.000001) GO TO 50
20 IF (X*Y .LT. 0.00001) GO TO 30
    X = X-Y-Y
30  X = X+Y
    ...
50 CONTINUE
    X = A
    Y = B-A
    GO TO 11
    ...
```



Similar structure may occur in assembly code

Historical Debate

- Dijkstra, Go To Statement Considered Harmful
  - Letter to Editor, *C ACM*, March 1968
- Knuth, Structured Prog. with go to Statements
  - You can use goto, but do so in structured way …
- Continued discussion
  - Welch, "GOTO (Considered Harmful)^n, n is Odd"
- General questions
  - Do syntactic rules force good programming style?
  - Can they help?

Advance in Computer Science

- Standard constructs that structure jumps

  if … then … else … end

  while … do … end

  for … { … }

  case …

- Modern style
  - Group code in logical blocks
  - Avoid explicit jumps except for function return
  - Cannot jump *into* middle of block or function body

Exception Concepts

- An exception is an unusual/unexpected/erroneous event in the program’s execution.
- An exception is “raised” when the event occurs.
- An exception is “thrown” when it is raised explicitly.
- An exception handler is a code segment that is executed when the corresponding exception is raised.

Exception Handler

- Example (in Ada):

```
loop
   ABLOCK:
     begin
       PUT_LINE (“Enter a number”);
       GET (NUMB);
     exit;
     exception
        when DATA_ERROR =>
        PUT_LINE (“Not number – try again”);
   end ABLOCK;
end loop;
```

Exception Handler in Java /C++

- Example (in Java):

```
try {
   …
} catch (Exception e){
   …
}
```

Continuation

- Where to continue execution after the exception handler?
  - The statement that raised the exception?
  - After the statement that raised the exception?
  - After the current iteration of a block? (Ada loop)
  - An explicit location?
  - At the end of the subprogram in which the exception was raised? (Ada)
  - After the exception handler? (Java/C++)
  - Nowhere – terminate the application? (unhandled exceptions)

Handler Selection

- Exceptions can be specified by:
  - Special exception type (Ada)
  - Ordinary data type (C++)
  - Object type with specified superclass (Java)
- Handler can be selected according to:
  - First match (Java/C++)
  - Best (most specific) match

First match

```
try {
  // can throw exceptions
} catch (Derived &d) {
  // Do something
} catch (Base &d) {
  // Do something else
} catch (...) {
  // Catch everything else
}
```

- Control jumps to first matching catch block
- Order matters if multiple possible matches
  - Especially with inheritance-related exception classes
  - Put more specific catch blocks before more general ones
  - Put catch blocks for more derived exception classes *before* catch blocks for their respective base classes
- **`catch(...)`**
  - catches any type

Exception Specifications C++

```
// can throw anything
void Foo::bar();


// promises not to throw
void Foo::bar() throw();


// promises to only throw int
void Foo::bar() throw(int);


// only char or int
void Foo::bar() throw(char,int);
```

- Make promises to the caller
- Allow stronger type checking enforced by the compiler
- By default, a function can throw anything it wants
- A throw clause in the signature
  - Limits what a function can throw
  - A promise to the calling function
- A throw clause with no types
  - Promises nothing will be thrown
- Can list multiple types
  - Comma separated

Exception Propagation

- If an exception is not handled by the subprogram in which it is generated, control is returned to the caller and the exception is reraised.
- If the main program has no handler, the program terminates.

Default Handlers

- Some languages have default handlers for some exceptions – Ada usually terminates the program.
- Generic handlers can be specified as a fallback mechanism:
- `catch (Exception e)` in Java
- `catch (…)` in C++
- `others`in Ada

finally

- Java has a special exception handler clause to be executed whether or not an exception occurred, and before control passes beyond the handler. Example:

```
try {
…
} catch (Exception e) {
…
} finally {
…
}
```

Summary

- Structured Programming
  - Go to considered harmful
- Exceptions
  - “structured” jumps that may return a value
  - dynamic scoping of exception handler