Subprograms

CS4700

Kenneth Sundberg

Subprogram

- Basic Subprograms have:
    - Single entry point
    - Suspend the caller

Subprogram

- Basic Subprograms have:
  - Single entry point
  - Suspend the caller
- Multiple entry points gives coroutines

Subprogram

- Basic Subprograms have:
  - Single entry point
  - Suspend the caller
- Multiple entry points gives coroutines
- Avoiding suspension gives concurrency

Definition

- The definition includes
  - Interface
  - Actions

Call and Return

- Call - the request to enter a sub program
- Return - the resumption of the calling program (possibly with a value
- A subprogram is active between call and return

Procedures and Functions

- Procedures
  - Do not return
  - They are intended as extension points for statements in the language
  - Mostly a feature of older languages
- Functions
  - Return
  - Modeled on math functions
  - Generally should not have side effects

Coroutines

- Include Yield and Resume
- Yield returns a value but maintains current state
- Resume restarts co-routine after last yeild
- Call and Return still exist and define the lifetime

Side effects

- Ways in which a CS function is not like a math function
- Context
  - global variables
  - static local variables
- Error
- Non-determinisim
- Destruction
  - I/O
  - out parameter

Referencing environments

- Set of bindings visible to a subprogram
  - local variables
  - nested subprograms

closures

- A subprogram and its referencing enviornment are called a closure

Return Values

- What are the types of return
- What are the number of return types

Formal and Actual

- The parameter definitions in the header are called formal
- The parameter values in a call site are called actual

Positional and Keyword

- If the matching between formal and actual parameters is based only on order then the language uses positional parameters
- If each actual parameter can be associated with a formal parameter name in any order the language used keyword parameters
  - foo(bar = 42)

Parameter passing

- pass by value
    - Only the value is passed (a copy)
- pass by result
    - A local variable is created
    - Value (result) is copied into caller at end of function
- pass by value result
    - Copy passed to function
    - Value copied back into caller
    - Also called pass by copy
- pass by reference
    - Create and copy an alias
- pass by name
    - As if parameter was textually substituted
    - Referencing environment must also be included for name lookups

Type checking parameters

- Do formal parameters have type?
- Do formal and actual parameters have to match?

multidimensional arrays as parameters

- A language needs to be able to build the array mapping
- This complicates passing arrays
  - Send a pointer and do pointer arithmetic
  - Less flexible functions (specific array size and layout)
  - More complex built in arrays

subprograms as parameters

- How can subprograms be passed - what restrictions are there?
- What is the referencing environment?
  - call statment - shallow binding
  - passed function definition - deep binding
  - specified at call site - ad hoc binding

Indirect subprograms

- Function pointers
- delegates
  - A Callable and assignable object
- virtual functions
  - Implemented in terms of indirect subprograms

Overloaded functions

- Subprograms with the same name and referencing environment
- Each must have a different protocol (number, type, and order of arguments)

Overloaded Operators

- Some languages (C++, Ada, Python, Ruby, others) allow operators to be overloaded
- Usually there is some special function name that is invoked by operator syntax

Generic subprograms

- Generic subprograms work on multiple types
- The concept of a parameter is what the generic subprogram expects
- A type is said to model the concept if it meets the requirements
- Generic programs work on all types that model their concept

Prologue and Epilogue

- Function call must
  - Suspend caller
  - Compute and pass parameters
  - Pass return address
  - Transfer control
- Return must
  - Resolve out parameters
  - Pass return value
  - Return control
  - Resum caller into previous state

Activation Records

- Data needed by every invocation of a function
- Stack local variables
- Parameters
- Return address
- Dynamic link
- Static link

Example: Recursive Factorial

Blocks

- Entering a block adds a new activation record
- Chains of static links used to lookup non-local names

Dynamic Scope

- Deep access - lookup names using dynamic links
- Shallow access - maintain a stack for each name
- Semantics are identical

Textbook sections covered:

- Section 09-02 (frame 2)
- Section 09-04 (frame 8)
- Section 09-05 (frame 13)
- Section 09-06 (frame 16)
- Section 09-07 (frame 17)
- Section 09-08 (frame 10)
- Section 09-08 (frame 10)
- Section 09-09 (frame 18)
- Section 09-10 (frame 20)

Textbook sections covered:

- Section 09-11 (frame 19)
- Section 09-12 (frame 9)
- Section 09-13 (frame 6)
- Section 10-03 (frame 22)
- Section 10-04 (frame 22)
- Section 10-05 (frame 24)
- Section 10-06 (frame 25)
- Section 10-2 (frame 21)