CS 351
Design of Large Programs
Programming Abstractions

August 30, 2021

Searching for the Right Abstraction

- The language we speak relates to the way we think.
- The way we view programming affects the kinds of systems we construct.
- Thus, the level of abstraction impacts:
  - Programming productivity
  - Reasoning about programs
  - Program analysis
  - Formal verification

Control Abstractions

Flow of control defines the order in which instructions are executed.

- *Sequential flow of control* is built into most machines (program counter)
- *Conditional jumps* allow the interpreter to skip and/or repeat code segments
- *if* and *goto* statements provide a more uniform treatment by separating the condition from the flow of control transfer.
- Further reductions in complexity are achieved by the shift to *structured programming*
- *Exceptions* provide a structured mechanism for handling error conditions

Procedural Abstractions

Procedural abstractions laid the foundation for modular design.

- *Macro substitution* offered a mechanism for naming sections of code and inserting them as needed
- *Subroutines* (non-recursive), introduced as a memory saving device, structured the flow of control
- *Blocks* provide an in-line structuring construct central to all modern programming languages
- *Procedures* (recursive) encapsulate processing, thus eliminating the need to look at the way they are coded (if a specification exists!)
- *Functions* (pure) are procedures without side effects

Data Abstraction

Data is central to the representation of state.

- *Built-in data types* provide essential but primitive forms of data representation and operations on them
- *Programmer-defined types* facilitate tailoring data selection to the specifics of the application
- *Statically Strongly-typed** languages improve dependability by doing strict compile-time checking

Data Abstraction Cont.

- *Abstract data type (ADT)* is a formal characterization of a set of data structures sharing a common set of operations
- *Generic types* (e.g., generics in Java) allow for parameterized definitions

Concurrency

Concurrency impacts the choice of components and connectors.

- *Coroutines* introduced the notion of passing the flow of control among subroutines
- *Concurrent process* (e.g., task in Ada, thread in Java) provides for logically parallel execution
- *Inter-process communication* assumes a variety of forms
  - Shared variables
  - Message passing
  - Remote procedure call
- *Synchronization* (mutual exclusion, barriers, etc.)

Functions Revisited

As in mathematics, a function defines a transformation from its inputs to the outputs.

- It has no side effects (no memory and no changes in the program state)
- It is deterministic (same inputs generate the same outputs every time)

Function Example: factorial

- factorial(n) where n is a natural number returns
  - 1 if $n = 0$
  - $n \cdot (n-1) \cdots 1$ if $n > 0$
- Functions are often defined recursively factorial(n) returns
  - 1 if $n = 0$
  - $n \cdot \mathrm{factorial}(n-1)$ if $n > 0$
- What happens if n is an integer? factorial(n) returns
  - error if $n < 0$
  - 1 if $n = 0$
  - $n \cdot \mathrm{factorial}(n-1)$ if $n > 0$

Factorial in Java

```
public static int factorial(int n)
    throws ArithmeticException {
  if(n < 0) throw new ArithmeticException();
  else if(n == 0 || n == 1) return 1;
  return n * factorial(n-1);
}
```

```
public static int factorial(int n) {
  if(n < 0) {
    System.err.println("Error: Factorial is undefined
                        + " for negative integers.");
    return 0;
  }
  else if(n == 0 || n == 1) return 1;
  return n * factorial(n-1);
}
```

Axiomatic Specification

A mathematical relation between the input and output values.
Assertions represent a convenient abstract mechanism for function specification

- An assertion is a logical fact that is true about the state of the program at some point in its execution
- Some programming languages provide assertions as built-in constructs
- A pre-assertion defines the relevant properties of the input values
- A post-assertion defines the relevant properties of the output value

Axiomatic Spec: Sort

Sort(X) returns Y

pre:

- X is an array of integers indexed from 0 to N

post:

- Y is an array of integers indexed from 0 to N
- Y is sorted in ascending order
- any integer k occurs the same number of times in both X and Y

Operational Specification (Pseudocode)

- An operational specification is an abstract program that:
  - Establishes the desired relation between inputs and outputs
  - Places no restrictions on how the function is ultimately coded
- Any code that accomplishes the same transformation is acceptable
- Some coding solutions may be more efficient than others

Pseudocode: Sort

Sort(X) returns Y
given:

- X is an array of integers indexed from 0 to N
- Y is an array of integers indexed from 0 to N

1. copy X into Y
2. while (there exists i and j such that $i < j$and $Y[i] > Y[j]$)
   - swap Y[i] and Y[j]

Sort in Java: bubblesort

```
public static int[] bubbleSort(int[] ary) {
  // assert ary.length > 0;
  int length = ary.length-1;
  boolean swap = true;

  while(swap) {
    swap = false;
    for(int i=0; i<length; i++) {
      if(ary[i+1] < ary[i]) {
        int tmp = ary[i];
        ary[i] = ary[j];
        ary[j] = tmp;
        swap = true;
      }
    }
  }
  return ary;
}
```

Sort in Java: quicksort

```
public static void quickSort(int[] ary, int low, int high) {
  if(ary == null || ary.length == 0) return;
  if(low >= high) return;

  int mid = low + (high - low) / 2;

  int pivot = ary[mid];
  int i = low, j = high;
  while(i <= j) {
    while(ary[i] < pivot) i++;
    while(ary[j] > pivot) j--;
    if(i <= j) {
      int tmp = ary[i];
      ary[i] = ary[j];
      ary[j] = tmp;
      i++;
      j--;
    }
  }
  if(low < j) quickSort(ary, low, j);
  if(high > i) quickSort(ary, i, high);
}
```

Procedures Revisited

Procedures, in contrast to functions, may have side effects due to:

- Local variables
- Access to resources
- Access to devices

The result of invoking a procedure may lead to

- Returning data whose values depend on the internal state of the procedure
- Changes in the internal state of the procedure

The specification methods are similar except for:

- The treatment of the internal state!

Abstract State Specification

- The internal state (e.g., data structures) of a procedure may be highly complex
- Proper abstraction of the internal state simplifies greatly the specification
- Users of the procedure need not be exposed to the internal data representation
- Internal representation may change over time
  - Specification is not affected
  - Code may be drastically affected

Documentation Implications (1/2)

Pre and Post assertions are the best way to document procedures and methods.

- Assertions are very helpful when placed at critical junctions in the code.

Pseudocode

- is not as helpful as assertions in documenting code
- is very good at capturing processing logic, e.g., explicit task scheduling
- must be highly abstract with a typical ratio of 1:10 (text vs. code)

Documentation Implications (2/2)

Focusing on an abstract state

- is challenging
- simplifies documentation
- protects the documentation against implementation changes
- is primarily associated with object and class documentation