Computer Programming I

Avery Karlin

Contents

Primary Textbook:

Teacher:

1 Introduction to OCaml

1.1 Program Design

1. Programming can be viewed as the division into four steps, understanding the program, formalizing the interface, writing test cases, and implementing behavior
   (a) The first involves finding any relevent concepts and how they relate to each other within the program, while the second determines input and output, and their respective formats
   (b) The third determines the correct answer to standard cases, edge cases, and erroneous cases (in which the user is at fault) for how the program should behave, writing test cases in the program
   (c) The fourth step is the programming, often taking recursive decompositions of the problem, dividing it by concept, the relationships between, and the interface to allow easier debugging

1.2 Basic OCaml Programming

1. OCaml supports integer (int), boolean (bool), and string (string) primitive types, able to be defined into expressions combining the types and operations on them, working by order of operations

(a) Basic operations on integers include +, -, *, /, mod, string$_of_int(convertingintegertostring), onstringsinclude$
$, <> (inequality), <, <=, >, >=$

2. Models of computation are ways of thinking about how a program executes, allowing prediction of its behavior, such as object or value oriented programming
   (a) Value-oriented is the idea that running expressions reduces it to a value, rather than an action, such as input/output, as the basis of OCaml
   (b) This is designated by the notation $< exp >=>< val >$, called the evalutation of the expression by calculation as an abstract model of the computer calculation
      i. The internal steps are written out by $\|- >< intermediaryexp >$ by order of operations
3. If-then-else constructs, written "if ¡exp¿ then ¡exp¿ else ¡exp¿", evaluating one of the last two based on the initial expression
   (a) Since the constructs are expressions by themselves, the else is required since the construct expression needs a value if the expression is false
4. OCaml programmings are made of declarations and commands, the former defining constants, functions, and expressions purely value oriented, the latter testing and generating output, non-value oriented
   (a) The let declaration is written as "let ¡id¿ = ¡exp¿", binding the identifier to the value, unable to be modified otherwise in the program
      i. It is often annotated by "let ¡id¿ : ¡type¿ = ¡exp¿" for ease of reading, though this is not required
      ii. Let declarations then automatically substitute each subsequent ¡id¿ with the value of the expression