Review

Concepts in Programming Languages

What we have learned

- Skills
  - Language syntax (context-free grammar, parse tree, and AST)
  - Lambda calculus (apply beta reduction)
  - Functional programming (recursion in Scheme and ML)
  - Type inference (from Scheme to ML)
  - Tail recursion, loops, and continuation passing (methods of programming)
  - Object-oriented programming (from ML datatype/abstype to C++ classes)
- Knowledge (concepts)
  - Language semantics (expressing power, interpretation vs. compilation, higher-order functions, functions as first-class objects)
  - Types, type checking and type inference; Polymorphism
  - Memory management (blocks, functions, classes and inheritance)
  - Continuation and exceptions
  - Abstractions, object-oriented abstractions,
  - C++ and Java language design and implementations
- Advanced topics
  - What if we modify a language by adding …

Skills

- Language syntax and context-free grammar
  - How to define a language using BNF?
  - Parse trees and abstract-syntax trees
  - Ambiguity of grammars (advanced topics)
    - Precedence and associativity; How to rewrite ambiguous production rules
- Lambda calculus
  - Understand the syntax and reduce to normal form
- Functional programming in Scheme and ML
  - Define recursive functions in Scheme and ML
- Type inference and translation between languages
  - What are the types of variables in a Scheme/ML code?
  - Translate Scheme code to ML
- Continuation passing, tail recursion, and loops
  - What is continuation passing? What is tail recursion? How to systematically convert program implementations?
- Object-oriented programming
  - Translate ML abstype/datatype/higher-order functions to C++ classes

Programming

- Programming is all about expressing things
  - using functions, alternatives, recursion, loops
- Exercise (going all the way)
  - Give a CFG for the syntax of regular expressions over {s,n}, where s and n stands for symbol and number respectively. For example
    - "s|n", "s*", (sn|ns)* are in the languages
    - "s|" and "*n" are not in the language
  - Give an example input in the language. Give parse tree and AST for the input. Rewrite your grammar to be non-ambiguous
  - Write a Scheme function that takes an AST of the RE, and returns how many symbols are inside the AST
  - Infer types of variables in you Scheme function. Define a ML datatype to represent the AST
  - Translate your Scheme function to ML; rewrite it to use continuation passing. Can you translate it to loops?
  - Translate your ML datatype and function to C++

Layout of C++ Class Objects

- Key: supporting dynamic binding of methods, subtype polymorphism, and class inheritance
- Exercise: draw the memory layout for the following classes
  - class A { private: int x;
    public: void foo() {…}
    virtual int bar(int z) {…} };
  - class B : public A
    { private: float y;
    public: void foo(float z) {…}
    virtual int bar(int z) {…} };
  - class C : public A
    { public: virtual int foo() {…} };

Blocks and Memory Management

- Key: understand the algorithm (get pass the syntax barrier)
  - Function definitions can be nested inside one another, but a function block is not entered untilled the function is invoked by a caller
- Exercise: list the order of events for the following code; then draw the runtime stack snapshot.

```
1: let
2:    fun mk_x(x) =
3:        let fun add1(y) = x + y
4:        in
5:            let val x = 7 in add1(5) end
6:        end
7:    fun apply(f,x) = f(x)
8: in
9:  apply(mk_x,10)-2
10:end;
```

Lambda Calculus

- Higher order functions to the extreme
  - Use functions to express everything
  - Key: understand function abstractions and function applications
- Exercise: apply beta reductions
  - λ x. (λ y. y x) (λ z. x z)
  - (λ x. (λ y. y x) (λ z. x z)) (λ y. y z)
  - (λ y. (λ x. λ y. x (x y)) (λ g. g y)) 5

Concepts: Languages and Functions

- Why high-level programming languages?
  - Productivity, portability, maintenability, machine efficiency
- What can programming languages express?
  - Data and algorithms
  - Partial recursive functions
- Programming paradigms
  - Can you define what they are and give examples?
    - Functional, imperative, object-oriented
  - What is a high-order function? What does "functions are first-class objects" mean?
- In what ways can prog. languages be implemented? Give examples? What is the trade-off? What are the implementation phases
  - Compilation vs. interpretation
  - Lexical analysis, Syntax analysis, semantic analysis, interpretation/code generation+optimziation

Concepts --- Types

- What is a type? What is it used for?
  - Types are classification of values
  - Different types of values have different layout/interpretation
- Type declaration and equivalence
  - Name vs. structure type equivalence
- What is a type system
  - How to determine types of variables and expressions?
    - Compile-time vs. runtime type checking
  - Type checking vs. type inference
    - Compile-time vs. runtime type checking
  - Type safety of languages
- Polymorphism
  - Parametric, ad-hoc and subtype polymorphism

Concepts --- Scopes and Runtime Control

- What is a block? Can blocks overlap with each other?
  - Block: a region of code that has local variables
- What is the scope and lifetime of a variable?
- What are local variables, global variables and function parameters?
  - Local variables: defined inside the current block
  - Global variables: defined in an enclosing block
  - Functions parameters: input and return parameters
- What is the scoping rule of a language?
  - Static scoping vs. dynamic scoping
- What is the memory model of program execution?
  - The memory model: runtime stack, heap, code space
  - Runtime stack:
    - Push an Activation record whenever encountering a new block
    - Environment pointer, control link, access link

Concepts-- Implementing Functions

- How many ways can parameter values be passed?
  - Pass by value vs. pass by reference
- What is a function closure? What is it used for?
  - The value of a function <code, env>
  - Used to setup environment for function calls
- Why is implementing higher-order functions hard?
  - When a function returns other functions, the activation records needs to be saved
  - Activation record in the heap ➔ OO languages
- What is tail recursion? Why is it equivalent to loops?
  - Tail recursion: do not need to return
- What is a continuation? What is continuation passing
  - Continuation: the rest of computation after function exit

Concepts: Exceptions

- Why are exceptions considered dynamic jumps?
  - Static jumps: goto, loop, conditionals, …
  - Exception:
    - Jump out of one or many levels of nested blocks
    - Until reaching some program point to continue
    - Pass information to the continuation point
- What is required from a language to support expections?
  - Type (exception) declaration
  - Raise an exception
  - Handle an exception
- Are exceptions part of the type system?
  - Raising of exceptions not part of type system
  - Handling of exceptions need to agree with type system

Abstractions

- What is abstraction?
  - Separate interface from implementation
  - Grouping of relevant data and functions
- How many ways can a language support abstractions?
  - Function/procedure abstraction
    - ML vs. C++/Java functions
    - Enforced by scoping rules
  - Data abstraction (encapsulation)
    - ML abstype, C++/Java classes
    - Enforced by type system
  - Modules: group of data and function abstractions
    - ML signatures and structures, C++ namespaces, C++/Java classes, Java interfaces
- Parameterization of abstractions (skipped)
  - C++ template

Object-oriented Abstractions

- OO abstractions are types
  - Have constructors and can be used to build objects
  - Grouping of relevant data and functions
  - Access control: private, protected, public, friend, package
- Encapsulation
  - Separate interface from implementation details
- Subtype polymorphism
  - Values of subtypes can be used to substitute base type values
- Dynamically-bound functions
  - Function pointers stored inside class objects
  - Virtual function are looked up at runtime
- Implementation inheritance
  - Derived classes can redefine virtual functions of base classes

Object-oriented languages

- C++/Java classes vs. ML datatype + scoping (nested functions)
  - ML can simulate most features of C++/Java except
    - Inheritance and extensibility
  - Java/C++ encapsulation ←→ ML function closure
  - Java/C++ namespaces ←→ML structures/signatures
  - Java/C++ virtual methods←→function pointers as values
  - Java/C++ subtyping ←→ union types and pattern matching
- Implementation of classes C++ vs. Java
  - Layout of class objects and Java interfaces
  - Managing class member functions
  - Design philosophies of the two languages