3342 Notes

**Closures:** functions that are stored inside other and that await to be executed.

do loop{

(do ((i 1 (+ 1 i)) (prod 1))  
 ((> i n) prod)  
 (set! Prod (+ i prod)  
}

**Scanning vs Parsing**

Scanning – **lexical analysis**: grouping characters together into token or words.

* The rules for the classes are language specific, and can usually be described by a “regular language”
* **Tools for scanners:** take the grammar for the regular language and produces a program.
  + Lex, flex, jflex

Parsing – **syntactic analysis**: grouping a liner sequence of tokens into a tree according to a set of rules.

* The rules for making the trees are language-specific, and can usually be described by a “context-free language”, ex. Well nested parentheses and arithmetic expressions.

**Character sets:** ASCII, EBCDIC, Latin 1, extension for ASCII, Unicode

**Compilers vs Interpreters**

Interpreter

* Takes a representation of a program and executes it.
* Maybe implemented in software, hardware, or a combination of both.
* Ex.
  + Byte-code interpreter in JVM
  + PowerPC instruction interpreter in hardware
* H = language being interpreted
* X = language in which the interpreter is implemented in.

Transformer

* Takes a representation of a program and creates an equivalent program.
* A transformer that transforms a program from a language to another is called a translator:
  + A translator that does a whole-program sophisticated analysis and produces the initial code in lower-level language is called a “compiler’
  + A translator that uses the same language for its input and output but improves the efficiency of the code is called an ‘optimizer’
* I = input language.
* O = output language.
* X = the language the transformer is implemented in.
* **Compilations phases**
  + Lexical Analysis
  + Syntactic Analysis
  + Semantic Analysis
  + Intermediate code generation
  + Intermediate code improvement
  + Machine-specific code generation
  + Machine-specific code improvement
* Translator/interpreter programs: IDE’s, optimizers, and dynamic compilation.

**Syntax Extension via Macros**

* Programmers can introduce their own syntax extension via macros.
* Used by, define-syntax, and syntax-rules.
* Macros are tree to tree transformations, therefore, miss-grouping is not possible
* Macros are simple and powerful in scheme compared to C
  + Scheme has a uniform representation for source trees
  + Transformations are based on trees
  + Different transformations can be applied in different situation as detrained by patterns and programs.
  + Macros are hygienic.

**Bindings and scopes**

Associating a name to a location is called binding.

Static scoping: lexical.

Dynamic scoping: fluid scoping.

* Deep binding: variable look-up is slow, context switch is fast.
* Shallow binding: variable look-up is fast, context switch is slow.

**C++**  
  
Methods are fields of a structure which act like functions and can access the other fields.

A class is a structure which has some of its fields declared private and others public.

* A class that is defined as an extension of another is called derived from.
* The first class is called a base class.
* A base class part is initialized with a call to its constructor following a “:” and before the body of the function.

Templates introduce a parameterized family of functions or classes, add by using <>

**Type Systems**

* Types are values that are stored in a computer as sequences of bits.
* Types provide an interpretation to binary data.
* Programming language differ in the way they associate types with data

**Static typing** is being able to bind data with one and only one type of data that has been declared to that variable name prior to the execution, also known as **strong typing**.

* Certain errors are caught earlier.
* Extra info => programs are more self-documenting, compilers have to do less testing, and therefore, more efficient code.

**Implementation**

* Type Inference: avoid type test and type storing, determine types at compile time.
* Allows overloaded operators, and overloaded resolution.

**Dynamic typing** is being able to bind multiple data types with the same variable name, also known as **week typing.**

* Small programs are easier to write.
* Some variables maybe used for different types, polymorphism.
* Some higher-order programs are impossible to type in statistical types.

**Implementation**

* Tagged Values: associate extra information with the value to determine the type.

**Object Oriented Programming OO**

* Midway between static and dynamic typing, allows the user to know some information about the object early, base class and methods.
* Detailed information is delayed until run time, exact derived class and method implementation.
* Object oriented types are represented as [ptr-to-class-info|Data Data Data …] the first part is a pointer to the class information, such as class name and methods tables (called vtbl virtual function table in C++), and the class data which is dynamically typed.
* Basics OO typing:-
  + Void: empty type, used for languages where a function must always return a value.
  + Singleton: a type with 1 element, nil, null, ‘()
  + Boolean: either byte or not a bit, special arrays of boolean will pack to bits.
  + Integer: signed, unsigned, short, normal, long.
  + Floating point number: IEEE standard is the most common, provides precision, different lengths.
  + Characters: ASCII, EBCDIC, Unicode, encoding is not determined by hardware.
  + String: primitive in some languages.
  + I/O ports or Files: wide variation on how they’re handled, provided by libraries, primitive in some languages.

**Constructors**

* Record/Struct: Heterogeneous
* Arrays: Homogeneous.
* Union: allows an object to store values of different types.
  + Tagged: a structure with a special discriminate field to indicate the correct interpretation for the data.
  + Untagged: up to the programmer to organize the data.
* Functions

**Mutable vs immutable types**

* Mutable: can be modified
* Immutable: cannot be modified
  + Can be stored in the ROM or shared without side effects.
  + Immutable is defined by const in C

**Subtypes**

* Denoted by S < T, S is a subtype of T

**XML**

W3C: World Wide Web Consortium

XML vs. HTML

* Case-sensitive
* Every opening <X> must have a closing </X>
* Empty <X></X> can be abbreviated <X/>
* Attributes must be quoted.
* Unicode.

XML is used for:

* Marking up data.
* Revision of HTML as an XML application: XHTML.

**XML Vocabulary**

* Document fragment: a piece of XML between an opening and closing pair.
  + Well-formed if, all pairs inside are properly nested.
  + Valid if, satisfies the XML grammar DTD (Document type description)
* Tags: the variables inside the angle brackets <>
* Element: a document fragment inside a particular tag.
* Entity: a short hand for a character or element, uses the syntax &Name.

**Namespaces**

* Used to avoid collisions between tags with the same names.
* Can be introduced using xmlns = XML Namespace.
* Setting an xmlns changes the default Namespace.

**XML Transformations**

* XSLT is a pattern-matching language for XML documents.

**Patterns**

* “/” matches on the root.
* “word” matches a <word> entity
* An extension of XPath
  + Long form syntax
  + Short form syntax

**XPath Expressions**

* Boolean, number, string, or a node-set. A node-set is an unordered collection of nodes with duplicates.
* Paths:
  + child::para selects the para element children of the context node.
  + child::\* selects all element children of the context node.
  + child::text() selects all text node children of the context node.
  + child::node() selects all the children of the context node, whatever their node type.

**Parameter Passing**

* **Call by value**: causes a copy of the value to be transmitted from the callee to the caller.
* **Multiple value return**
  + Global variables
  + Language support
* **Call by reference**: passes an address to where the data is stored.
* **Call by Value-Result:** all the benefits of call by reference, clear semantics.
  + Original value copied to the function, then returned at the end of the program.
* **The sneaky Loop-Hole:** passes two arguments, the first is passed by value, second is passed by reference.
  + Efficient way to pass big objects.
  + A trap/feature, the object can be updated by the callee.