## Control Flow

The *control flow* of a program is the way an execution moves from statement to statement.

The textbook breaks it down into:

- Sequencing (do the next thing)
- Selection (pick something to do, e.g. if, switch)
- Iteration (repeat something, e.g. while, for)
- Recursion

Unit 10

SI 413 Control Flow

Structured Programming

Generics

Software Licenses

• Unstructured (e.g. goto)

Unit 10 SI 413	Unstructured flow: GOTO
Control Flow	
GOTO	
Structured Programming	
Generators	In the beginning, there was GOTO. And GOTO was good.
Generics	
Software Licenses	<ul> <li>Directly jumps from one place (the goto) to another (the label)</li> </ul>
	<ul> <li>Corresponds exactly to machine code</li> </ul>
	Very efficient
	Can cause some problems

Unit 10 SI 413	Good Use of Goto?
Control Flow GOTO	Say we want to print a vector with commas like "1, 2, 3".
Structured Programming	This solution prints an extra comma!
Generators Generics Software Licenses	<pre>vector <int> v; // int i = 0; while (i &lt; v.size()) { cout &lt;&lt; v[i] &lt;&lt; ",_"; ++i; } cout &lt;&lt; endl;</int></pre>

Unit 10 SI 413	Goto Problems
Control Flow	
GOTO	
Structured Programming Generators Generics Software Licenses	<ul> <li>They don't play well with scopes. (Restricting to local gotos avoids this.)</li> <li>Can be used to cook up "spaghetti code" — hard to follow.</li> <li>Hard to know where we are in the program, i.e., hard to reason about the program's correctness/performance.</li> </ul>

Unit 10	int x = 0;
SI 413	char c;
51 413	goto rs;
Control Flow	fns:
	if (c != '1' && c != '0') goto er;
GOTO	goto ns;
Structured	rd:
Programming	c = getchar();
Generators	ns:
	if (c == '1') { x = x*2 + 1; goto rd; }
Generics	if (c == '0') { x = x*2; goto rd; }
Software	es:
Licenses	if (c == '_')
	{
	c = getchar();
	goto es;
	}
	if (c == '\n') goto done;
	er:
	<pre>printf("Error!\n");</pre>
	return 1;
	rs:
	c=getchar();
	if (c == '_') goto rs;
	else goto fns;
	done:
	printf("%i $n$ ,x);

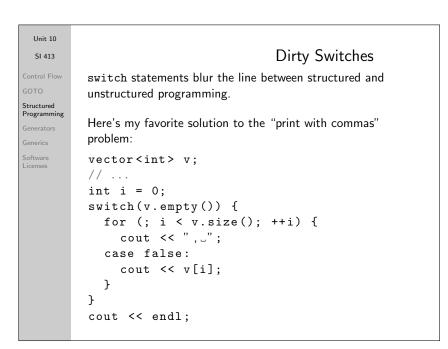
Unit 10 SI 413 Control Flow	Structured Programming
Structured Programming Generators Generics Software Licenses	<i>Structured programming</i> is probably all you have ever known. Championed by Dijkstra in the pioneering paper "GOTO Statement Considered Harmful" (1968).
	Structured programming uses control structures such as functions, if, while, for, etc., even though these are mostly compiled into gotos. Allows us to reason about programs, enforce modularity, write bigger and better programs.

Unit 10	
SI 413	Looping over a Collection
Control Flow	
GOTO	
Structured Programming	
Generators	
Generics	How would you write $C++$ code to loop over the elements of
Software Licenses	• an array?
	• a linked list?
	• a binary search tree?
	How can we separate interface from implementation?

Unit 10 SI 413	Iterators
Control Flow	
GOTO	
Structured Programming Generators Generics Software Licenses	<ul> <li>An <i>iterator</i> needs to be able to:</li> <li>Get initialized over a collection.</li> <li>Move forward (maybe backwards?) through a collection.</li> <li>Fetch the current element</li> <li>Know when it's done.</li> </ul>
	In C++, an iterator overrides ++ and $*$ to become an abstract pointer.
	In most other languages (e.g., Java), an iterator has to extend an abstract base type with next() and hasNext() methods.

Unit 10 SI 413	For-Each Loops
Control Flow GOTO Structured Programming	A <i>for-each loop</i> provides an even easier way to loop over the elements of a collection.
Generators Generics Software Licenses	<pre>Java example: HashSet <string> hs; // for (String s : hs) { System.out.println(s); // This prints out all the strings in the HashSet. }</string></pre>

This construct is supported by most modern languages. Often there is a direct connection with iterators. In some languages (e.g., Python), this is the *only* for loop.

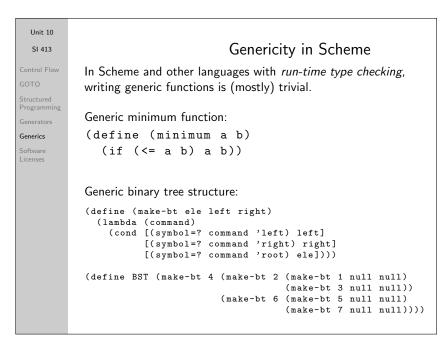


Unit 10	
SI 413	Aside: Scripting Languages
Control Flow	
GOTO	bash, Ruby, Python, Pearl, and PHP are examples of
Structured Programming	scripting languges.
Generators	They are designed for <i>small tasks</i> that involve coordination or
Generics	communication with other programs.
Software Licenses	Common properties:
	<ul> <li>Interpreted, with dynamic typing</li> </ul>
	• Emphasis on <i>expressivity</i> and <i>ease of programming</i> over efficiency
	<ul> <li>Allows multiple paradigms (functional, imperative, object-oriented)</li> </ul>
	• Built-in string handling, data types
	• Extensive "shortcut" syntactic constructs

```
Unit 10
                             Scripting example: Prime
  SI 413
                                  generation in Python
Control Flow
Structured
Programming
Generators
          def PrimeGen():
Generics
Software
Licenses
             for p in itertools.count(2):
                if all(p%i != 0 for i in range(2,p)):
                  yield p
          for p in PrimeGen():
             if p < 100: print(p)
             else: break
```

## Unit 10 SI 413 Control Flow GOTO Structured Programming Generators Generators Sometimes a function computes multiple values as it goes along. An iterator created automatically from such a function is called a generator Simpler (related) Python example: def factors(n): for i in range(2,n): if n % i == 0: yield i

Unit 10	
SI 413	The Need for Generic Code
Control Flow	
GOTO	
Structured Programming	
Generators	A <i>function</i> is an abstraction of similar behavior with <i>different</i>
Generics	values.
Software Licenses	<i>Generic</i> code takes this to the next level, by abstracting similar functions (or classes) with <i>different types</i> .
	Most common usages: • Basic functions: min/max, sorting
	• Collections: vector, linked list, hash table, etc.

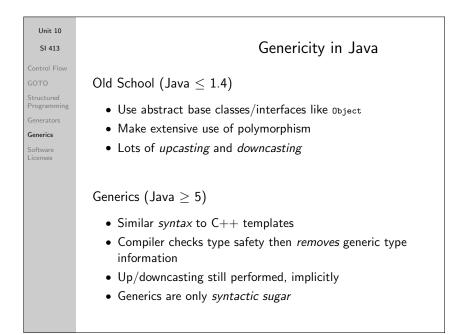


## Unit 10 SI 413 Control Flow GOTO Structurd Programming Generators Generics Software Licenses Old School (C style) • Use function-like macros to code-generate every possibility. • Types to be used in generic functions/classes must be explicitly specified. Templates (C++ style) • Built into the language; don't rely on preprocessor

- Compiler does code generation, similar to macros
- Types to be used are determined *implicitly* at compile-time
- Separate compilation becomes difficult or impossible.

```
#define WRITE_LL_CLASS(T) \
class Node_ ## T { \
  public: \
     T data; ∖
     Node_ ## T * next; \setminus
     Node_ ## T (T d, Node_ ## T * n) :data(d), next(n) { } \
١
     T printAndSum() { \
       cout << data << endl; \setminus
       if (next == NULL) return data; \setminus
       else return data + next->printAndSum(); \
     } \
};
WRITE_LL_CLASS(float)
WRITE_LL_CLASS(int)
int main() {
  Node_float* fhead = NULL;
  Node_int* ihead = NULL;
  // ... fill the lists with some input
  cout << "Floating_sum:_" << fhead->printAndSum() << endl << endl;
cout << "Int_sum:_" << ihead->printAndSum() << endl << endl;</pre>
}
```

```
template <class T>
class Node {
  public:
    T data:
    Node <T> * next;
    Node<T> (T d, Node<T> * n) :data(d), next(n) { }
    T printAndSum() {
      cout << data << endl;</pre>
      if (next == NULL) return data;
      else return data + next->printAndSum();
    7
};
int main() {
  Node<float>* fhead = NULL;
  Node<int>* ihead = NULL;
  // ... fill the lists with some input
  cout << "Floating_sum:_" << fhead->printAndSum() << endl << endl;</pre>
  cout << "Int_sum:_" << ihead->printAndSum() << endl << endl;</pre>
  return 0;
}
```



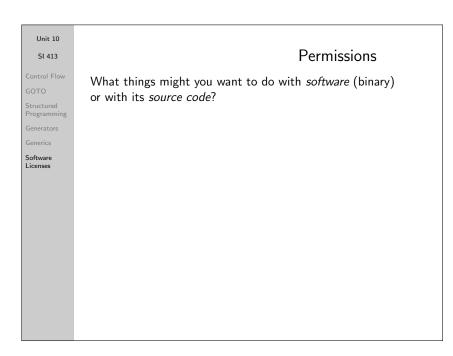
Unit 10	
SI 413	Manual Genericity in Java
Control Flow	
GOTO	
Structured Programming	<pre>interface Sum { void add(Number x); }</pre>
Generators	class FloatSum implements Sum {
Generics	float val = 0;
Software Licenses	<pre>public void add(Number x) { val += ((Float)x).floatValue(); } public String toString() { return String.valueOf(val); }</pre>
	}
	<pre>class IntSum implements Sum {     int val = 0;     public void add(Number x)     { val += ((Integer)x).intValue(); }     public String toString() { return String.valueOf(val); } }</pre>

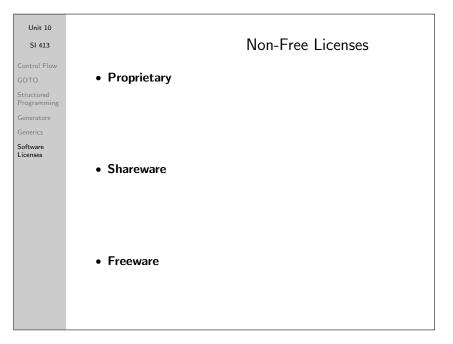
```
Unit 10
             class LLO1d {
  SI 413
                Number data;
                LLOld next;
                LLOld(Number d, LLOld n) { data = d; next = n; }
                Sum printAndSum(Sum summer) {
Programming
                  System.out.println(data);
                  summer.add(data);
Generics
                  if (next != null) next.printAndSum(summer);
Software
Licenses
                  return summer;
                }
                public static void main(String[] args) {
                  LLO1d flist = null;
                  LLO1d ilist = null;
                   // ... fill the lists with some input
                  System.out.println("Floating_sum:_" +
  flist.printAndSum(new FloatSum()) + "\n");
System.out.println("Integer_sum:_" +
                     ilist.printAndSum(new IntSum()) + "\n");
                }
             }
```

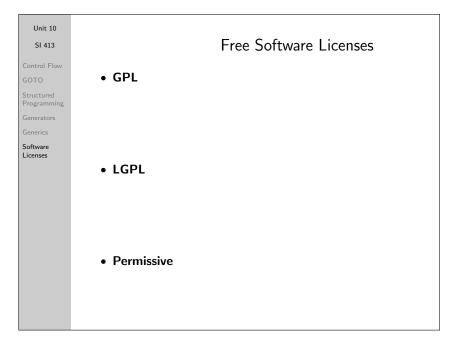
```
Unit 10
                                             Java 5 Generics
  SI 413
Structured
            interface Sum<T> { void add(T x); }
Programming
            class FloatSum implements Sum<Float> {
Generics
             float val = 0:
              public void add(Float x)
Software
Licenses
              { val += x.floatValue(); }
             public String toString() { return String.valueOf(val); }
           ŀ
           class IntSum implements Sum<Integer> {
             int val = 0;
             public void add(Integer x)
              { val += x.intValue(); }
             public String toString() { return String.valueOf(val); }
           }
```

```
Unit 10
            class LLNew<T> {
  SI 413
              T data:
              LLNew<T> next:
              LLNew(T d, LLNew<T> n) { data = d; next = n; }
Structured
              Sum<T> printAndSum(Sum<T> summer) {
Programming
                System.out.println(data);
                summer.add(data);
Generics
                if (next != null) next.printAndSum(summer);
Software
Licenses
                return summer;
              7
              public static void main(String[] args) {
                LLNew<Float> flist = null;
                LLNew<Integer> ilist = null;
                // ... fill the lists with some input
                \tt System.out.println("Floating\_sum:\_" + \\
                  flist.printAndSum(new FloatSum()) + "\n");
                System.out.println("Integer_sum:_" +
                  ilist.printAndSum(new IntSum()) + "\n");
              }
            }
```

Unit 10 Trade-Offs in Generics SI 413 Programming No declared types • No enforced notion of "list of integers" etc. Generics Requires dynamic typing; slower Software Licenses • **Code Generation** (C++ templates) • Can result in (combinatorial!) code explosion • Very powerful and general, but somewhat unintuitive • Code Annotation (Java 5 generics) · Syntactic sugar; extensive run-time casting results • Types not known to the program at runtime eliminates some capabilities







Unit 10 SI 413	Case Studies
SI 413 Control Flow GOTO Structured Programming Generators Generics Software Licenses	Case Studies • Template programming and the LGPL • Linux on the Kindle • Iceweasel • Vernor v. Autodesk • "Badgeware" and CPAL

Unit 10	
SI 413	Class outcomes
ontrol Flow	
ото	You should know:
tructured rogramming	• What structured vs unstructured programmings is.
enerators	<ul> <li>The goods and bads of GOTOs</li> </ul>
ienerics oftware	• What an iterator is, and where/how/why they are used.
icenses	<ul> <li>What a for-each loop is, and where/how/why they are used.</li> </ul>
	What a scripting language is
	What a generator is
	<ul> <li>What a generic class/function is</li> </ul>
	• How genericity works in C++ and Java
	Major types of software licenses
	<ul> <li>What copyleft is and why it matters</li> </ul>

SI Contr

Struct Progra Genera Generi Softwa Licens