Lecture 14: Control Flow

COMP 524 Programming Language Concepts
Stephen Olivier
March 19, 2009

Based on slides/notes by A. Block, N. Fisher, F. Hernandez-Campos, and D. Stotts

The University of North Carolina at Chapel Hill

Goal of Talk

- The goal of this talk is to talk about the flow of programs

Control Flow

- **Control flow** is the order in which a program executes.
- For imperative languages (e.g., Java), this is fundamental.
- For other programing paradigms (e.g., functional), the compilers/interpreters take care of ordering.

Control Flow Mechanisms

- Sequencing
  - Textual order, precedence in Expression
- Selection
- Iteration
- Procedural abstraction
- Recursion
- Concurrency
- Nondeterminacy

Sequencing

- **Sequencing** is the order in which statements are to be executed.
- For imperative languages, typically things are executed **in the order they appear**!

This is not necessarily the case for functional languages!

Selection

- **Selection** occurs whenever there is a choice between two or more courses of action.
    - e.g. if/then/else & switch/case.

If-Then-Else

- For complex conditionals two ways to evaluate
  - Evaluate and put into register (works but slow)
  - Use short-circuiting in assembly to have **jump codes** (fast and awesome)

If-Then-

```
if ((A>B) && (C>D))) or (E!=F)
then {then_clause}
else {else _clause}
```

Regular

```
  r1:=A
  r2:=B
  r1:=r1>r2
  r2:=C
  r3:=D
  r2:=r2>r3
  r1:=r1&r2
  r2:=E
  r3:=F
  r2:=r2!=r3
  r1:=r1|r2
  if r1=0 goto L2
L1: then_clause
  goto L3
L2: else_clause
L3:
```

Short Circuit

```
  r1:=A
  r2:=B
  if r1<=r2 goto L4
  r1:=C
  r2:=D
  if r1>r2 goto L1
L4: r1:=E
  r2:=F
  if r1 = r2 goto L2
L1:then_clause
  goto L3
L2:else_clause
L3:
```

Switch-Case

- Not only is it more convenient in certain circumstances but it is more efficient!
  - Can implement a case-switch as an indexed table rather than a very long piece assembly code.

Unstructured Flow: The GOTO statement

- Assembly languages controls flow via conditional and unconditional jumps

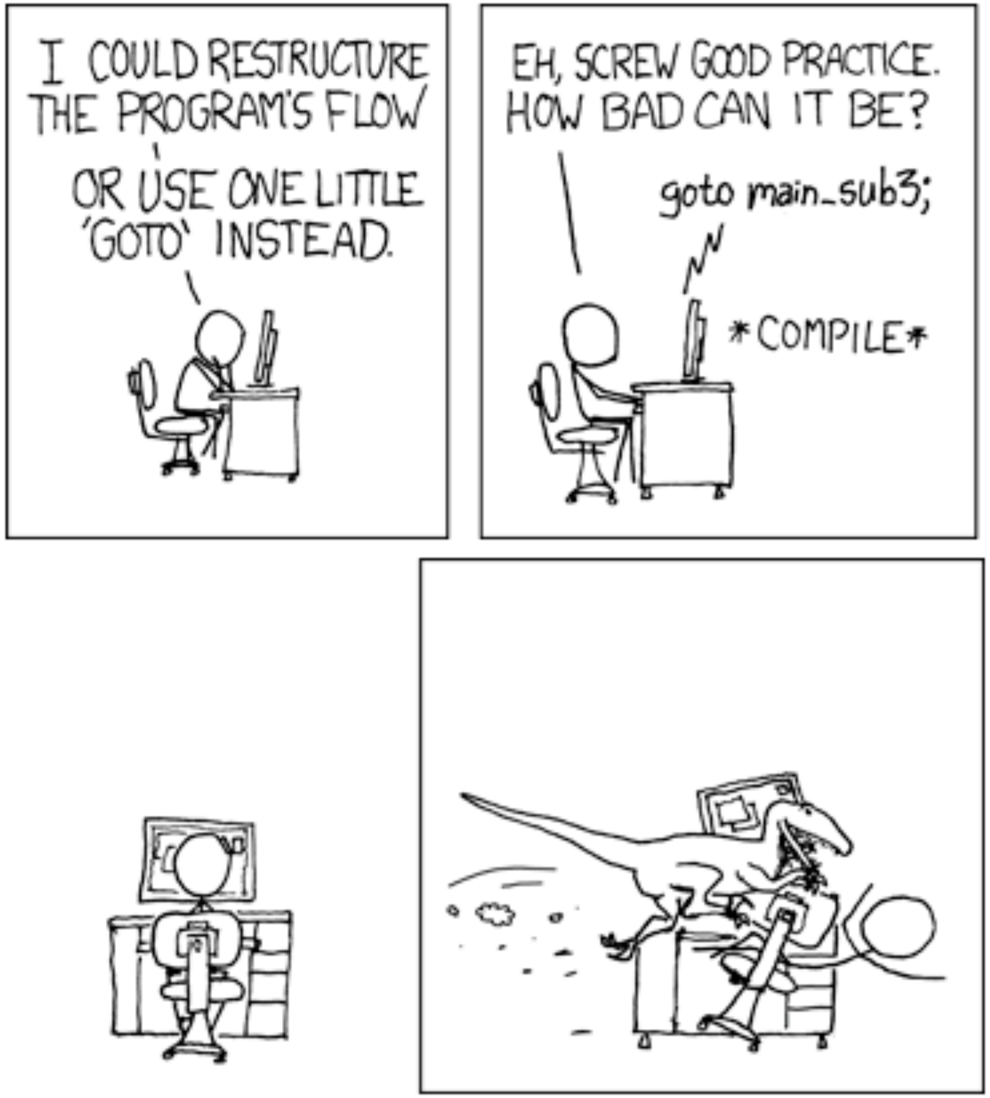
```
JMP 30
...
30:ADD r1, #3
```

Unstructured Flow: The GOTO statement

- Some higher level languages have similar statement

```
goto stop_point;
...
stop_point:
cout<<“stopping”;
```

Unstructured Flow: The GOTO statement



Unstructured Flow: The GOTO statement

- Using goto has long been considered bad practice
  - See “Goto Considered Harmful” paper
  - “Spaghetti code”
  - Difficult to debug

Structured Flow

- Structured flow (i.e., if-then-else, loops, etc...) provide the same expressive power
  - Bohm & Jacopini in 1964 proved that sequencing, selection, and itteration can effectively emulate gotos
- However, sometimes gotos are more convenient.

Special Cases--Perl, redo

```
while ($d++){
  #redo jumps to here
  $r = random($d);
  if($r>100) {redo};
  $sum +=$r*$d;
}
```

Special Cases--Perl, last

```
while ($d++){
  if($d>=37) {$res = “done”; last;}
  $sum +=$d;
}
#last jumps to here
```

- Similar effect in C/C++/Java with `break` statement

Special Cases--Perl, next

```
while ($d<37){
  $d++;
  if(($d%5)==1) {next};
  $sum +=$d;
  #next jumps to here
}
```

- Similar effect in C/C++/Java with `continue` statement

Special Cases

- Early subroutine returns

```
void ncaaRound2(String team) {
  if (team == “Dook”) {
    cout << “Better luck next year”;
    return;
  }
  ncaaRegionals(team);
}
```

- Exceptions and Errors

Iteration and Recursion

- These two control flow mechanisms allow a computer to perform the same set of operations repeatedly
  - Otherwise program code size is linear to the amount of computation to be done!
  - Also, needed to be able to express any algorithm
    - We call all language that can do this **Turing complete**
- **Functional languages** mainly rely on **recursion**.
  - We discussed its use in ML
- **Imperative languages** mainly rely on **iteration.**

Iteration

- Iteration usually takes the form of loops
- Two principal varieties:
  - **Enumeration** controlled loops: iterates through an **enumerated set**.
  - **Logically** controlled loops: iterates while (or until) a **logical statement is true**.

Examples

```
for (int i =0;i<=10;i++){
  ...
}
```

```
int i = 0;
while (i<=10){
  ...
  i++;
}
```

Enumeration

Logical

Iteration: Enumeration-Controlled Loops

- **Fortran** enumeration-controlled loops are comprised of several elements
  - **Label at end of loop**
  - **Index variable**
  - **Bounds and step size**
  - **Body of the loop**

```
 do 10 i=1, 100, 2
   ...
10:continue  ! no-op
```

Problems with Fortran

- Loop **boundaries** must be integer
- Index variable can **change within body of loop**
- **Goto statements** may **jump** in and out of loop
- The value of **i after termination** of the loop is **implementation dependent**
- The test of the loop **takes place at the end** so **body is executed at least once**.

```
 do 10 i=1, 10,2
   ...
10:continue
```

Iteration: Empty conditions

```
FOR i:= first TO last BY step DO
  ...
END
```

```
   r1:=first
   r2:=step
   r3:=last
L1: if r1>r2 goto L2
   ...
   r1:=r1+r2
   goto L1
L2:
```

```
   r1:=first
   r2:=step
   r3:=last
   goto L2
L1: ...
   r1:=r1+r2
L2: if r1<=r3 goto L1
```

Ite … ditions

Slow

```
…t TO last BY step DO
```

```
   r1:=first
   r2:=step
   r3:=last
L1: if r1>r2 goto L2
   ...
   r1:=r1+r2
   goto L1
…2:
```

```
   r1:=first
   r2:=step
   r3:=last
   goto L2
L1: ...
   r1:=r1+r2
L2: if r1<=r3 goto L1
```

Only works if

$first+(\lfloor(last-first)/step\rfloor+1)step$

is at most the largest integer.

```
  ...
END
```

```
   r1:=first
   r2:=step
   r3:=last
L1: if r1>r2 goto L2
   ...
   r1:=r1+r2
   goto L1
L2:
```

```
   r1:=first
   r2:=step
   r3:=last
   goto L2
L1: ...
   r1:=r1+r2
L2: if r1<=r3 goto L1
```

Backwards loop

- Decrement rather than increment the index variable
- Some languages have an explicit notation:

```
FOR i:= last DOWNTO first BY step DO
```

Access to Index Outside the Loop

```
var c: ‘a’ .. ‘z’;
FOR c:= ‘a’ to TO ‘z’ DO
BEGIN
 ....
END;
```

```
   r1:= ‘a’
   r2:= ‘z’
   if r1>r2 goto L3
L1: ...
   if r1=r2 goto L2
   r1:=r1 +1
   goto L1
L2: c:=r1
L3:
```

Access to Index Outside the Loop

Preserves c after loop

```
var c: 'a' .. 'z';
FOR c:= 'a' to TO 'z' DO
BEGIN
  ....
END;
```

```
    r1:= 'a'
    r2:= 'z'
    if r1>r2 goto L3
L1: ...
    if r1=r2 goto L2
    r1:=r1 +1
    goto L1
L2: c:=r1
L3:
```

Iteration: Iterators

- **Iterators** are used to **enumerate the elements of any well-defined set**.
  - Moreover, they **generalize arithmetic sequences**.
- In previous examples, iteration was always over the **elements of an arithmetic sequences**

```
for i in int$from_to_by(first,last,step) do
  ...
end
```

Clu

foreach in Perl

```
@colors = (“red”, “green”, “blue”)
foreach $elt (@colors){
  print $elt, “, ”;
}
print “are the colors we have\n”;
```

```
@colors = (“red”, “green”, “blue”)
foreach (@colors){ #use $_
  print $_, “, ”;
}
print “are the colors we have\n”;
```

Iterators as objects

- Java allows for **iterators as objects**

```
hasNext(); // return true if next element
```

```
next(); // Returns next element
```

```
remove(); // Gets rid of the last element (optional)
```

Iteration: Logically-controlled Loops

- Three types:
  - **Post-test**: Test at end
  - **Midtest**: Test in middle
  - **Pre-test**: Test at beginning

Examples

```
repeat
  ...
until i==true;
```

```
for(;;){
  ...
  if i==true break;
  ...
}
```

```
while (i==false)
{
  ...
}
```

Post-test

Midtest

Pre-test

Parallel Loops

```
for(i = 0; i < 100; i++)
{
  C[i] = A[i] + B[i];
}
```

Processor 1

Processor 2

```
for(i = 0; i < 50; i++)
{
  C[i] = A[i] + B[i];
}
```

```
for(i = 50; i < 100; i++)
{
  C[i] = A[i] + B[i];
}
```

Parallel Loops

```
for(i = 0; i < 100; i++)
{
  C[i] = A[i] + B[i];
```

First 50 iterations

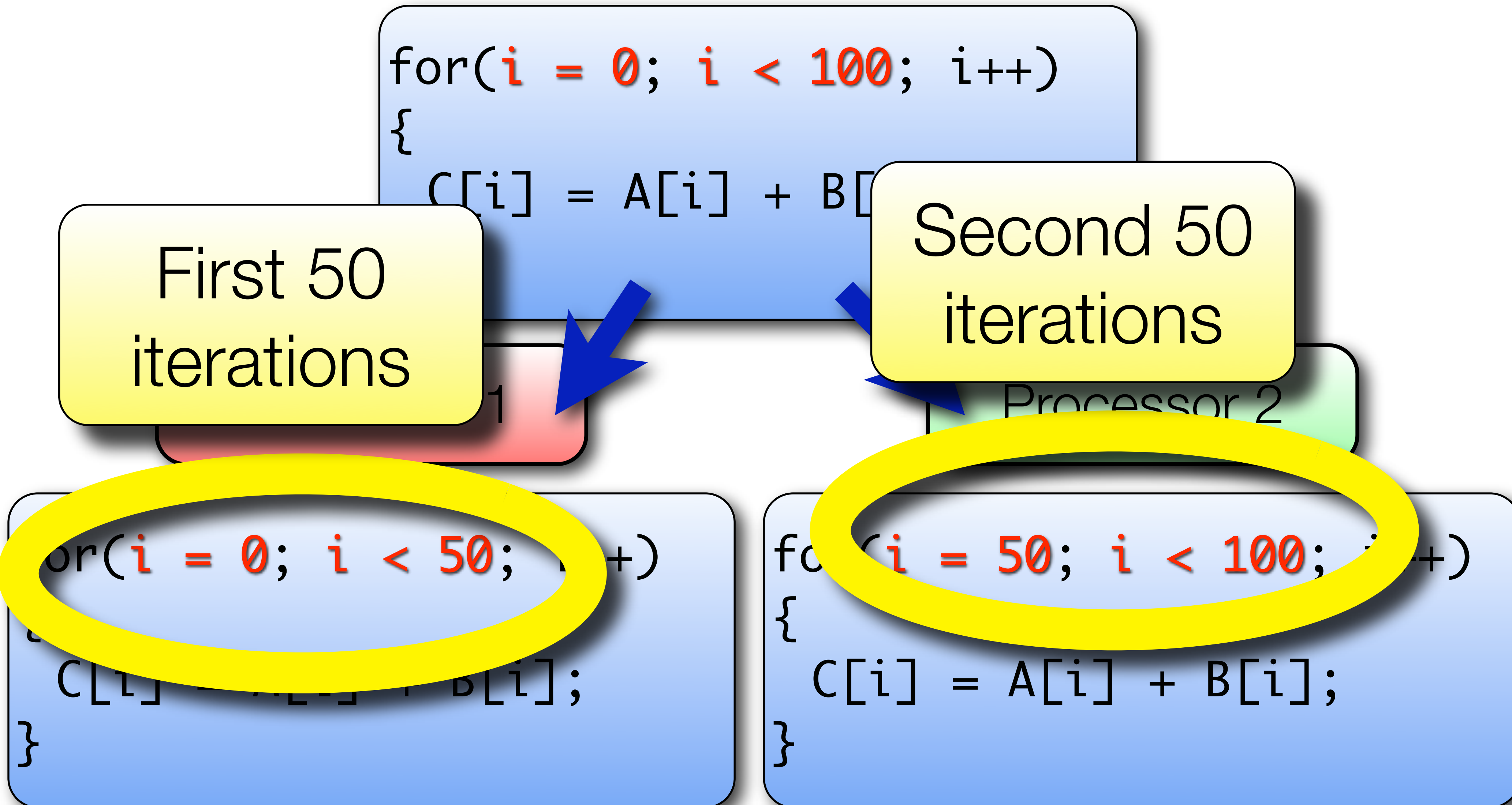
1

Processor 2

```
or(i = 0; i < 50; i++)
{
  C[i] = A[i] + B[i];
}
```

```
for(i = 50; i < 100; i++)
{
  C[i] = A[i] + B[i];
}
```

Parallel Loops



Parallel Loops

```
for(i = 0; i < 100; i++)
{
  grandtotal += A[i];
}
```

Processor 1

Processor 2

```
for(i = 0; i < 50; i++)
{
  grandtotal += A[i];
}
```

```
for(i = 50; i < 100; i++)
{
  grandtotal += A[i];
}
```

Parallel Loops

Concurrent update problem!

We will discuss options to fix this in lectures on concurrency.

Proc… …cessor 2

```
for(i = 0; i < 50; i++)
{
  grandtotal += A[i];
}
```

```
for(i = 50; i < 100; i++)
{
  grandtotal += A[i];
}
```