Programming Paradigms

Control Flow (Part 3)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Summer 2020
Overview

- Expression Evaluation
- Structured and Unstructured Control Flow
- Selection
- Iteration
- Recursion
Selection

- **Branch** that depends on a condition
- **Different syntactic variants**
  - If-else statements (sometimes with else-if)
  - Case/switch statements
If Statements

Syntactic variants across PLs

Algol 60 and its descendants:

if (A == B) then ...
else if (A == C) then ...
else ...

Lisp and its descendants:

(cond
((= A B)
(...))
((= A C)
(...))
(T
(...))
)

Bash

If [ $A = $B ]
then ...
elif [ $A = $C ]
then ...
else ...
fi
Compilation of If Statements

if ((A > B) and (C > D)) or (E \neq F) then
    then-clause
else
    else-clause

short-circuited evaluation
fall-through to some cases

r1 := A
r2 := B
if r1 \leq 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-branch
L3:
Case/Switch Statements

Many conditions that compare the same expression to different compile-time constants

--- Ada syntax
case ... -- potentially complicated expression
if
  when 1    => clause_A
  when 2 | 7 => clause_B
  when 3..5 => clause_C
  when 10   => clause_D
  when others => clause_E
end case;
Case/Switch Statements

Many conditions that compare the same expression to different compile-time constants

-- Ada syntax

```ada
case ... -- potentially complicated expression
if
  when 1 => clause_A
  when 2 | 7 => clause_B
  when 3..5 => clause_C
  when 10 => clause_D
  when others => clause_E
end case;
```

Labels          Arms

1               clause_A
2 | 7             clause_B
3..5             clause_C
10               clause_D
others           clause_E
Compilation of Case/Switch Statements

r1 = ... (calculate controlling expr.)

if r1 ≠ 1  goto L1

Clause-A
  goto L6

L1:  if r1 = 2  goto L2
     if r1 ≠ 7  goto L3

L2:  clause-B
     goto L6

L3:  if r1 < 3  goto L4
     if r1 > 5  goto L4
     clause-C
     goto L6

L4:  if r1 ≠ 10  goto L5
     clause-D
     goto L6

L5:  clause-E

L6:

disadvantage:
  linear pass through different cases
Jump-table-based Compilation

T: & L1 (expression = 1)
   & L2
   & L3
   & L3
   & L3
   & L5
   & L2
   & L5
   & L5
   & L4 (expression = 10)

L6: \texttt{r1 := ...} (evaluate expr.)
    if \texttt{r1 < 1} goto \texttt{L5}
    if \texttt{r1 > 10} goto \texttt{L5}
    \texttt{r1 := r1 - 1}
    \texttt{r1 := T [r1]}
    goto * r1

advantage:
constant-time jump
to right arm
Variations Across PLs

- Case/switch varies across PLs
  - What values are allowed in labels?
  - Are ranges allowed?
  - Do you need a default arm?
  - What happens if the value does not match?
C/C++/Java

- Each expression needs its own label (no ranges)
- Control flow “falls through”, unless stopped by `break` statement

```c
switch ( /* expression */ ) {
    case 1: clause_A
        break;
    case 2:
    case 7: clause_B
        break;
    case 3:
    case 4:
    case 5: clause_C
        break;
    case 10: clause_D
        break;
    default: clause_E
        break;
}
```
Quiz: Switch/Case

What does the following C++ code print?

```cpp
int x = 3;
switch (x)
{
    case 1: { x += x; } 
    case 3: { x += x; } 
    case 5: { x += x; } 
    default: { x += 5; } 
}
std::cout << x;
```

Please vote in Ilias
Quiz: Switch/Case

What does the following C++ code print?

```cpp
int x = 3;
switch (x)
{
    case 1: { x += x; }
    case 3: { x += x; }
    case 5: { x += x; }
    default: { x += 5; }
}
std::cout << x;
```

Result: 17

Please vote in Ilias