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 ...
```

Bash

```
If [ $A = $B ]
then ...
elif [ $A = $C ]
then ...
else ...
fi
```

Lisp and its descendants:

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

Compilation of If Statements

if ((A > B) and (C > D)) or
(E ≠ F) then

then_clause

else

else_clause

short-circuited evaluation

fall-through to some cases

```
     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_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
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**

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)
                    (evaluate expr.)
L6:  r1 := ...
     if r1 < 1  goto L5   } (L5 stores clause_E)
     if r1 > 10 goto L5   }
     r1 := r1 - 1
     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?

Fall-Through Case/Switch

C/C++/Java

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

```
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?

```
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?

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

Each of these is executed (because no `break` statement)

Result: 17

Please vote in Ilias