* Control Flow
- Selection - Dangling-else, Short-circuiting of conditional exp, multi-way selectors
- Iteration: logically controlled loops, combination/enumeration loops, anatomy, issues

Oct 22

Control Flow: Selection (if-statements)

Lab#2 is posted
Assignment 3 is posted.

C, C++, Java

```
if (<condition>) {
        <statement-1>
}
else {
        <statement-2>
}
```

optional

Plus, if <statement> is a single statement the braces are optional

Python

```
if <condition>:
      <statement-1>
else:
      <statement-2>
```

Also, for nested if-s

```
if <condition-1>:
      <statement-1>
elif <condition-2>:
      <statement-2>
...
else:
      <statement-N>
```

Dangling-else Problem

```
if <c1> if <c2> <s1> else <s2>
```

or

```
if <c1>
if <c2> s1 else <s2>
```

Two interpretations

```
①  if <c1>
   |     if <c2>
   |         <s1>
   else
       <s2>
```

```
②  if <c1>
       if <c2>
       |     <s1>
       else
           <s2>
```

Resolution in C, C++, Java

dangling else associates with the closest unmatched if — i.e. ② is the correct interpretation.

Either interpretation can also be forced using curly braces.

what about Python??

Short-circuited conditional evaluation

C

$$\text{int } x[N]; \qquad \overset{\text{Java}}{\text{int}[\,]\ x = \text{new int}[N];}$$

$$\text{while/if } (\overbrace{i < N}^{A}\ \&\&\ \overbrace{x[i] > 0}^{B})\ \{$$

$$\}$$

when $i = N$ (i.e. A is false), $x[i] > 0$ (i.e. B) is never evaluated. ~~[illegible]~~ Since the expression A&&B will always be false if A is false. I.e. A && B is short-circuited.

Q: What would happen if C/Java did not use short-circuiting??

A: when $i = N$ (i.e. A is false), it would still evaluate B (i.e. $x[N] > 0$). This would result in a run-time error in Java (ArrayIndexOutOfBoundsException).

In C, there is no bounds checking so it will examine some memory after x[] + compare it with Ø.

Short-circuiting ~~[illegible]~~ applies to all boolean expressions. C, C++, Java, Python do short-circuiting.

Q- What ~~[illegible]~~ would happen if

① A||B and A is false?

② A && B and A is true?

Applies to conditions in if- + while- + for-statements.

Multi-Way Selection: More than 2 conditions.

e.g. Given a date d/m/y e.g. 22/10/2024
compute # days in month, m in year, y.

C, C++, Java

```
if (m==2) {          // February
    if (leapYear(y))
        days=29;
    else
        days=28;
}
else if (m==1 || m==3 || m==5 || m==7
        || m==8 || m==10 || m==12)

        days=31;
    else if (m==4 || m==6 || m==9 || m==11)
        days=30;
        else {// ERROR...
             }
```

Python

```
if m==2:
    if leapYear(y):
        days=29
    else:
        days=28
elif m==1 or m==3 or ... or m==12:
    days=31
elif m in [4,6,9,11]:      #another way to test
                           # better than the one above)
    days=30
else: # ERROR...
```

C, C++, Java have a switch-case statement.

```
switch (m) {
    case 2 : if (leapYear(y))
                  days = 29;
             else
                  days = 28;
              break;
   case 3 :
    case 5 :
    case 7 :
     case 8 :
     case 10 :
      case 12 : days = 31;
                 break;
      case 4 :
       case 6 :
       case 9 :
        case 11 : days = 30;
                   break;
       default : { //ERROR...
                  |
                  }
}
```

```
Ada  case  m  is
          when  2  =>  ---
          when  3|5|7|8|10|12  =>  days = 31;
          when  4|6|9|11  =>  days = 30;
          when  others  =>  ...ERROR...
     end case;
```

Python: match-case (see Lab#2 handout)

```
match m:                    ← OR pattern
   case 1|3|5|7|8|10|12:
        days = 31
   case 4|6|9|11:
        days = 30
   case 2:
        if leapYear(y):
            days = 29
        else
            days = 28
   case _: # ERROR ...
```

Notes: C/C++: case values must be constants/literals of int or char type

Java: case values must be constants/literals and can be of int, char, boolean, or String type.

Also, constant expressions are allowed as case values.

Python: case value/pattern can be a simple value, a variable, or a more complex structure.

e.g.: or pattern (see above)

, use if-condition. e.g.

```
match n:
   case n if n>0: ___
   case n if n <0: ___
   case _: ___
```

complex structures can be lists, dictionary, etc. can get complex. See Python Reference.

Control Flow: Iteration/Loops

C, c++, Java

1. Logically Controlled Loops
   Have a loop condition and a loop body

~~2 king~~ 2 kinds: while and do-while

```
pre-test
while (<condition>) {              do {
    <statements>                       <statements>
}                                  } while (<condition>);
                                                   post-test
```

2. Combination/Enumeration Loops

```
for (<initialize>; <condition>; <update>) {
        <statements>
}
```

These are equivalent to the for-loop

```
<initialization>
while (<condition>) {
    <statements>
    <update>
}
```

Python

```
while <condition>:                 for <variable> in <sequence>:
    <statements>                       <statements>
```

Anatomy of a loop

1. Loop index / Loop control variable
2. Loop condition
3. Loop update
4. Loop body.

```
                         initialization   condition     update
e.g.   for (int i = 0 ; i < n ; i++) {
            <statements>  <------ body
       }
```

Also
c, c++, Java + Python

- break — exits the loop
- continue — skips current iteration + goes to next.

Other loop designs

c, c++, Java

```
while ( 1 ) {
      =
      if (<condition>)
          break;
      =
}
```

Python

```
while True :
      =
      if <condition> :
          break
```

c, c++, Java

```
for ( ; ; ) {
      =
      if (<condition>)
          break;
      =
}
```

These are written as infinite loops

Also, in Java

```
~~int ta[ ] x = new int [N];~~   int x[ ] = new int [N];

for (int a : x) {                 Same as
   — do something with a —        for (int i = 0, i < N, i++) {
}                                    — do something with x[i]
                                  }
```

Other languages

```
for i:=0 to n-1 do
begin
    <statements>
end
```

PASCAL

```
for i:=n-1 downto ∅ do
begin <statements>
end
```

Modula2

```
FOR i:=0 TO n-1 DO
    <statements>
END
```

```
FOR i=n-1 . TO ∅ BY -1 DO
    <statements>
END
```

Loop Design Issues

⊙ Loop control variable < not required in while-loops / also optional in for-loops

⊙ Scope of LCV

```
int i;
for (i=0;i<N; i++){
   =
}
// i is visible here
```

```
for (int i=0; i<N; i++){
   =
}
//i is NOT visible here
```

⊙ Can the LCV be modified in the loop body??

C,C++, Java

```
for (int i=0; i<10; i++){
    printf("—",i)
    i=i+1;
}
```

Python

```
for i in range(10):
    print(i)
    i=i+1
```

What is printed? Try it!