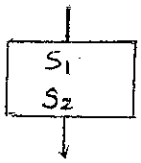
Control Structures (Constructs)

Bohm and Jacopini (1965)

Basic control structures: ***sequence, selection, iteration***

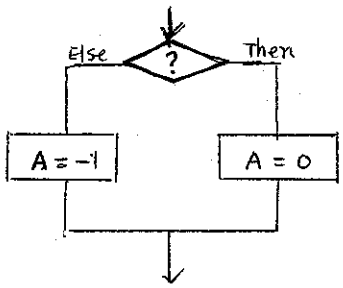
(1) Sequence (Compound)

S1
S2
:



(2) Selection (Conditional)

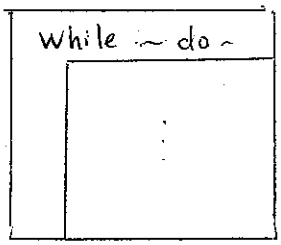
if (cond) then S1 else S2
if (cond) then S



(3) Iteration (Repetition)

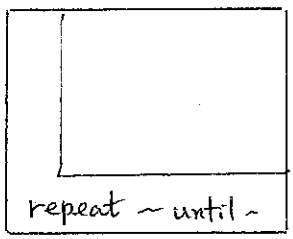
pretest

while (cond) do ~



posttest

repeat ~ until (cond)



Note. At least one execution. (Fortran)

Note.

At the lowest level (object code), goto (branch) is implemented.
Structured Programming: no 'goto'

Goto Controversy

Edsger Dijkstra (1965)
"Go To Statement Considered Harmful"
Comm. of the ACM, 11(3), 1968, 147-148

Frank Rubin (1987)
""Go To Statement Considered Harmful" Considered Harmful"
Comm. of the ACM, 30(3), 1987, 195-196

When 'goto' makes sense:

1.

```
for i:= 1 to n do
   if A[i] = 0 then goto next
   process A[i]
end
next:
```

```
flag:= false;
i:= 1
while (not flag) and (i <= n) do
   if A[i] = 0 then flag:= true;
   if (not flag) then process A[i]
   i:= i + 1;
end
```

2.

```
loop
   read(x)
   if x = 0 then goto next;
   process x
end
next:
```

```
read(x)
while x <> 0 do
    process x
    read(x)
end
```

3.

```
READ IN-FILE AT END GOTO EOF-RTN.
```

Selection

(1) Two-way Selection

if cond then S1 [else S2]

- Dangling-Else Problem

```
if (sum == 0)
    if (count == 0)
        result = 0;

else
    result = 1;
```

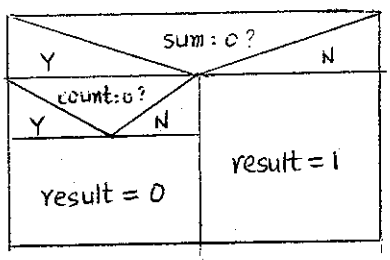
① Python. Indentation

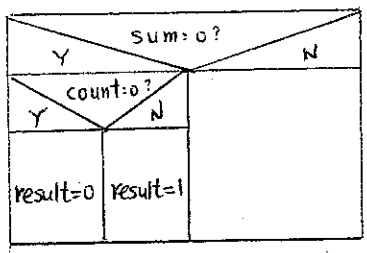
② C-based, Pascal. Nearest then

③ Perl. Each clause is compound.

```
if (sum == 0) {
  if (count == 0) {
     result = 0
  }
} else {
     result = 1;
}
```

```
if (sum == 0) {
  if (count == 0) {
     result = 0
  }
  else {
     result = 1
  }
}
```





④ Fortran, Ada, Ruby

```
if sum == 0 then
  if count == 0 then
    result = 0
  end
else
   result = 1
end
```

```
if sum == 0 then
  if count == 0 then
     result = 0
  else
     result = 1
  end
end
```

(2) Three-way Selection

Fortran. Arithmetic IF

```
      D = B**2 – 4 * A * C
      IF (D)  10, 20, 30
10    ...          (negative case)
20    ...          (zero case)
30    ...          (positive case)
```

Note. CAS instruction on IBM 704

(3) Multi-way Selection

Switch

C/C++, Java, JavaScript

```
switch (index) {                    // index = 1,2,3,4
  case 1:
  case 3:  odd += 1;
           sumodd += index;
           break;
  case 2:
  case 4:  even += 1;
           sumeven += index;
           break;
  default: printf("Error. Index = %d\n", undex);
}
```

C#

```
switch (value) {
    case -1:
        Negatives++;
        break;
    case 0:
        Zeros++;
        goto case 1;
    case 1:
        Positives++;
        break;
    default:
        Console.WriteLine("Error in switch \n);
```

Iterative Statement

for-loop, while-do, repeat-until

(1) Counter-Controlled Loop

Fortran

```
      DO  50   I = 1,20,2
         A(I) = …
50    CONTINUE
```

Fortran 90

```
      DO I = 1,20,2
         A(I) = …
      END DO
```

Ada

```
count :  Float := 3.14;          // count is a variable
for count in 1..10 loop          // count is a loop variable
  sum := sum + count;
end loop;
```

C-based

```
for  (count = 1;   count <= 10;  count++) {
        loop body
}
```

Note.

```
for (count1 = 0, count2 = 1.0;
     count1 <= 10 && count2 <= 100.0;
     sum = ++count1 + count2, count2  *= 2.5);
```

C99/C#

```
for (int count = 0; count < len; count++) {…}
```

Python.

```
for count in [2, 4, 6] :
    print count                          // 2, 4, 6

for count in range (5, 11, 2) :
    print count                          //  5, 7, 9
```

(2) Logically Controlled Loop

C/C++, Java

```
while (cond)                    do
    loop body                       loop body
                                while (cond)
```

- pretest - -posttest-

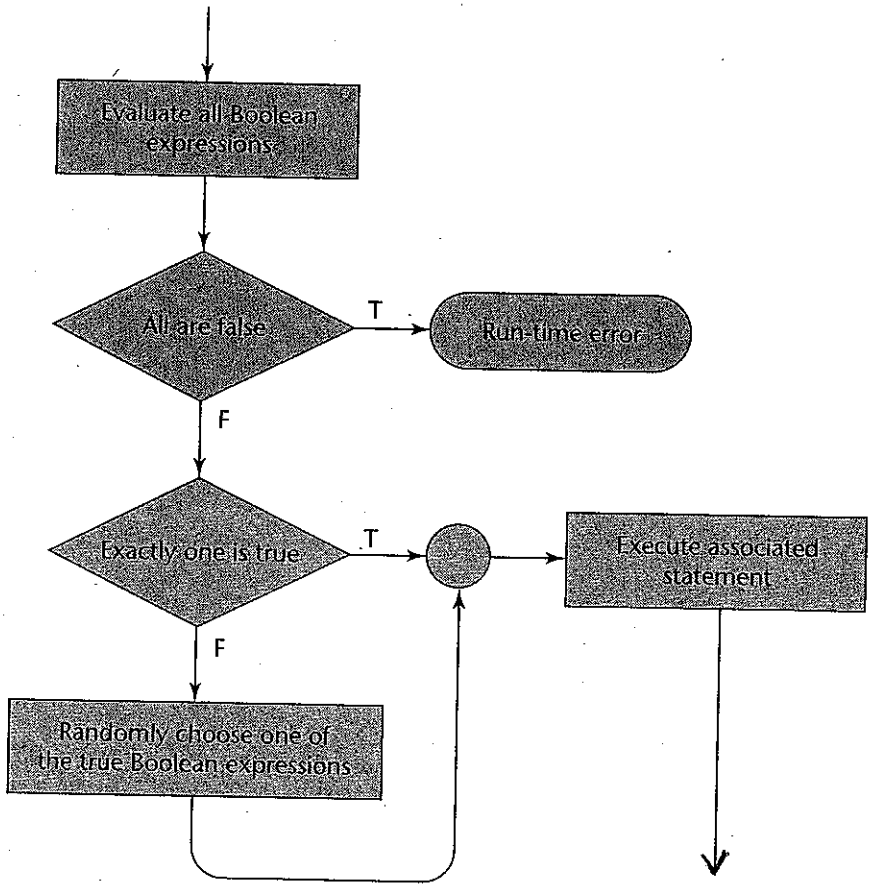
Guarded Command

- Dijkstra (1975)
- Concurrent programming - nondeterminism

Ex 1.

```
if   i = 0 -> sum := sum + i
[ ]  i > j -> sum := sum + j
[ ]  j > i -> sum := sum + i
fi
```

If (i = 0 and j = 2) ➔ choose 1st or 3rd statement nondeterministically.



Ex 2. Sort four integers, q1, q2, q3, q4.

```
do q1 > q2 -> temp := q1; q1 := q2; q2 := temp;
[] q2 > q3 -> temp := q2; q2 := q3; q3 := temp;
[] q3 > q4 -> temp := q3; q3 := q4; q4 := temp;
od
```