```
if (cond)
{
    Block
}
Stmt.
```

← Control flow

Premise: Given: Control flow reaches to Stmt.

We cannot conclude anything about cond!

Premise: Given: Control flow reaches to Block.

Conclusion: $Cond_{if} \equiv True$.

```
if (a > b)
{
    Block
}
```

[Control flow reaches to block $\longrightarrow a > b$] $\equiv$ True.

```
if (a > b && c < d)
{
    Block.
}
```

Control flow reaches to block $\longrightarrow (a > b \wedge c < d)$

$p \wedge q \equiv T \rightarrow p \equiv T$

$p \wedge q \equiv T \rightarrow q \equiv T.$

Control flow reaches to Block $\rightarrow a > b.$

Control flow reaches to Block $\rightarrow c < d.$

```
if(a < b || c == d)
{
    Block
}.
```

Control flow reaches to Block $\longrightarrow (a<b) \vee (c=d)$

Either $a < b$

or $c = d$

or both.

End of if-Stmt exercises.

```
if (cond)
{
    Block_if
}
else
{
    Block_else.
}
```

Control flow reaches to Block_if → Cond ≡ True.

Control flow reaches to Block_else → Cond ≡ False → ¬Cond ≡ True

---

```
if (a > b)
{
    Bif
}
else
{
    Belse
}
```

C.F. @ Bif → (a > b) ≡ True.

C.F. @ Belse → (a > b) ≡ False

$a > b \equiv$ False

$\neg (a > b) \equiv \neg$ False $\equiv$ True.

$\neg (a > b) \equiv$ True

$a \leq b \equiv$ True.

CoF. @ $B_{else} \rightarrow a \leq b$

De Morgan's law.

let p be a proposition.

let q be a proposition.

not (p and q) $\equiv$ (not p) or (not q).

p	q	p and q	not (p and q)
F	F	F	T
F	T	F	T
T	F	F	T
T	T	T	F

p	q	not p	not q	(not p) or (not q)
F	F	T	T	T
F	T	T	F	T
T	F	F	T	T
T	T	F	F	F

not (p and q) ≡ (not p) or (not q)

$\neg(p \wedge q) \equiv \neg p \vee \neg q$. — (1)

---

not (p or q) ≡ (not p) and (not q)

$\neg(p \vee q) \equiv \neg p \wedge \neg q$.

p	q	p∨q	¬(p∨q)	¬p	¬q	¬p∧¬q.
F	F	F	T	T	T	T
F	T	T	F	T	F	F
T	F	T	F	F	T	F
T	T	T	F	F	F	F

$\neg(p \vee q) \equiv \neg p \wedge \neg q$.

If else examp #2.

```
if (a >= b && c < d)
{
    B if
}
else
{
    B else
}.
```

C.F. @ $B_{if} \rightarrow (a \geq b) \wedge (c < d)$

C.F. @ $B_{else} \rightarrow [(a \geq b) \wedge (c < d)] \equiv F$

$\neg [\underbrace{(a \geq b)}_{p} \wedge \underbrace{(c < d)}_{q}] \equiv T.$

Apply De Morgan.

$\neg((a \geq b) \wedge (c < d)) \equiv \neg(a \geq b) \vee \neg(c < d)$

$\equiv (a < b) \vee (c \geq d)$

C.F. @ Belse:

Either $a < b$

OR $c \geq d$

OR Both.

```
if((a <= b) || (c != d))
{
    Bif
}
else
{
    Belse
}
```

C.F. reaches to Bif $\rightarrow$ $a \leq b \vee c \neq d$.

C.F. reaches to Belse $\rightarrow$ $\neg(a \leq b \vee c \neq d)$

$\equiv \neg(a \leq b) \wedge \neg(c \neq d)$

$\equiv (a > b) \wedge (c = d)$

ALG: find maximum of two numbers a & b.

```
int max(int a, int b)
{ int m;

    if (a > b)
        m = a;
    else
        m = b;
    return (m);
}
```

P. M.

Control flow reaches two Bif.

→ a > b.

Execute m = a.

a > b → a is max amongst a & b

State of var m = state of var a.

∴ State of var m = State of max. of a & b.

C.F. reaches to Belse → a > b ≡ F.

a > b ≡ False

e.g. $\neg(a > b) \equiv$ True.

$a \leq b \equiv$ True.

$a \leq b \equiv a < b$ or $a = b$.

In both cases b is max of a & b.

e.g. Execute (m=b)

→ State of var m = State of var b

∴ State var m is max. of a&b

```
void search(int *pa, int N, int Sd)
{
    int i;
    for(i=0; i<N; ++i)
        if(pa[i] == Sd)
            break;
    if(i<N)
        printf("Data found");
    else
        printf("Data not found");
}
```

Algorithm terminates ⟶
Control flow has come out of for loop.

C.F. out for loop ⟶
loop cond False or break executed.

Possibility #1:
loop condition false.

$i < N \equiv$ False.

$\neg(i < N) \equiv$ True.

$i \geq N \equiv$ True.

$i == N \equiv$ True ⟵ Conclusion-1

loop. condition ≡ False.

for values of i from 0 to N-1
(pa[i] == sd) ≡ False
⟶ How?

because if for any i_0 between 0 to N-1 ($0 \le i_0 \le N-1$) $pa[i_0] == sd$ had been true then break statement would have got executed with value of $i = i_0$ which is $< N$.

$\therefore i_0 < N$ & $i_0 = N$ cannot be satisfied together.

for $0 \le i < N$, $pa[i] == sd$ is false
$\Rightarrow$ search data not found.

Exited from loop because loop cond is false

$\rightarrow$ 1) $i = N$ just after exit from loop
2) Search data not present!

Possibility #2: break statement executed.

break statement executed ⟶
pa[i] == sd is true.
⟶ $0 \le i < N$

break stmt executed ⟶

$0 \le i < N$ ∧ data found in array

```
void insert_at_sorting_pos(int *pa, int N)
{ /* Precondition: N >= 2.
      pa[0] to pa[N-2] is sorted
      in ascending order
*/
  int tmp = pa[N-i];
  int k;
  k = N-2;
  while(k>=0 && pa[k]>tmp)
  {
```

```
        pa[k+1] = pa[k];
        k = k-1;
    }
    pa[k+1] = tmp;
    /* Post: pa[0] to pa[N-1] is
             sorted in ascending
             order
    */
}
```

Control flow out of while loop.
loop condition ≡ False.

$k \geq 0 \wedge pa[k] > tmp \equiv F$

$\neg(k \geq 0 \wedge pa[k] > tmp) \equiv T$

$\neg(k \geq 0) \vee \neg(pa[k] > tmp)$

$k < 0 \vee pa[k] \leq tmp$

Case #1: $k < 0$.

As we are decrementing k from N-2 by value of 1, $\boxed{k = -1}$

the loop terminated for $k = -1$

⟶

the loop body executed for values of k from N-2 to 0.

$k = N-2, N-3, N-4, --, 0.$

⟶

pa[k] > tmp is true for

$k = N-2, N-3, --, 0.$

Execution of Stmt.

pa[k+1] = pa[k]

for values $k = N-2, N-3, --, 0$

resp.

⟶

Numbers in array from index = 0 to index = N-2 at the start of ALG have been shifted to index = 1 to index = N-1 without changing internal order.

Precondition ≡ pa[0] to pa[N-2] sorted.

We just proved above that these elements are shifted to pa[1] to pa[N-1] without changing relative order.

i.e. at the end of loop pa[1] to pa[N-1] are sorted!

and

pa[1] to pa[N-1] > tmp.

After loop pa[k+1] = tmp.

pa[-1+1] = tmp

pa[0] = tmp

$= pa[N-1]$

start also.

$pa[0]$ to $pa[N-1]$ is sorted.

Case #2:

$k >= 0$ and $pa[k] > tmp$

$\equiv$ $\equiv$

True False.

Ans: 1: $0 \leq k \leq N-2$.

Ans 2: let k_0 be a number between 0 to $N-2$ ($0 \leq k_0 \leq N-2$) for which $pa[k_0] > tmp$ is False.

the loop terminated for k_0

$\longrightarrow$

loop executed for $k = N-2$ downto $k = k_0+1$

$\longrightarrow$ Array elements from $k = k_0+1$ to

k = N-2 > tmp.

loop body executes for
k = N-2 down to k = k0+1

→

at the end of loop
pa[k0+1] to pa[N-2]
are shifted to
pa[k0+2] to pa[N-1]
without changing internal order.

As pa[0] to pa[N-2] were sorted
pa[k0+2] to pa[N-1] are sorted.
pa[k0+2 : N-1] > tmp.

---

pa[k0] ≤ tmp.

As pa[0] to pa[N-2] were sorted
pa[0] to pa[k0] are also sorted

$pa[ks] \leq tmp.$

$pa[0 : ks] \leq tmp.$

$pa[0 : k_0] \leq tmp$ &

$pa[k_0+2 : N-1] > tmp.$

Execution $pa[ks+1] = tmp$

$\longrightarrow$

$pa[0]$ to $pa[N-1]$ is sorted. □

Systematic consideration

hidden