UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

CONTROL STRUCTURES - SELECTION

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

CONTROL STRUCTURES

So far we executed our programs top to bottom, left to right, line by line.

Control structure is an element that influence the current point of execution

> Sequence

> Selection (sometimes called 'alternation')

> Repetition (sometimes called 'iteration')

Control structures can conditionally execute a compound statement, also known as a block.

```
{
    . . .
}
```

C variables have block scope – a variable can only be accessed inside the block it was defined in.

CONDITIONAL OPERATORS

Compare two values

`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`==`	equality (Note: two equal signs, not one)
`!=`	inequality

The only issue is, be careful when comparing `float` (or `double`) types with `int` types

LOGICAL OPERATORS

Conditional operators are often used together with logical operators.

`&&`	logical 'and'
`||`	logical 'or'
`!`	logical 'not'

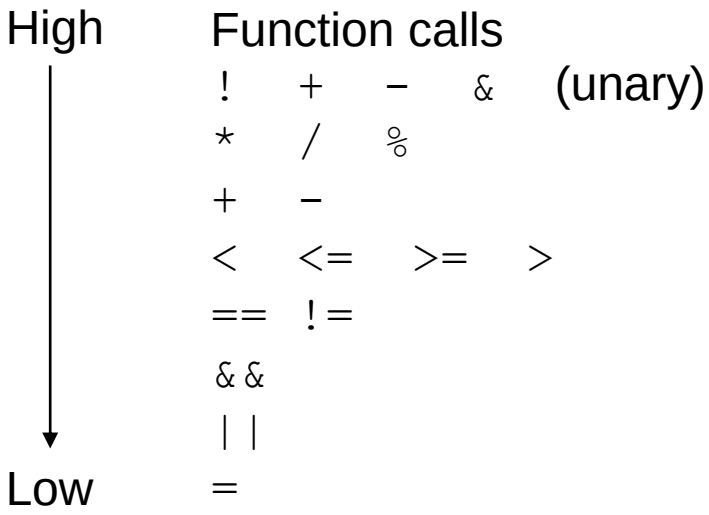
Can be used to combine results of multiple comparisons.
e.g. To cross the road check right **AND** left.

In C logical TRUE is represented with 1, logical FALSE with 0.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

ORDER OF OPERATIONS



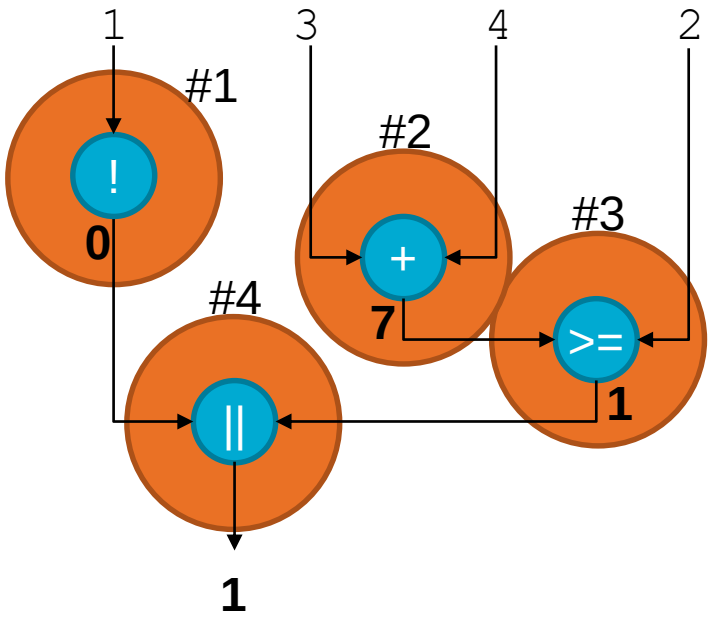
Short-Circuit Evaluation

When the truth/falsehood of a conditional statement can be determined, part the way through evaluation then the rest of the expression is ignored.

> This is a reasonably obvious optimisation measure

EXAMPLE 1

```
int flag=1; int x=3; int y=4; int z=2;

!flag || (x+y >= z)
```



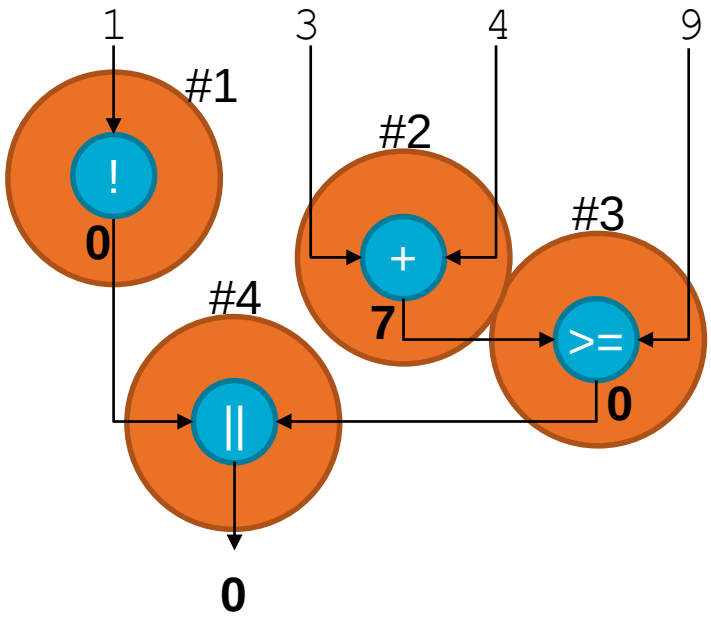
Result: TRUE

Compare/manipulate two values at a time only.

The tree structure is called the evaluation tree.

EXAMPLE 2

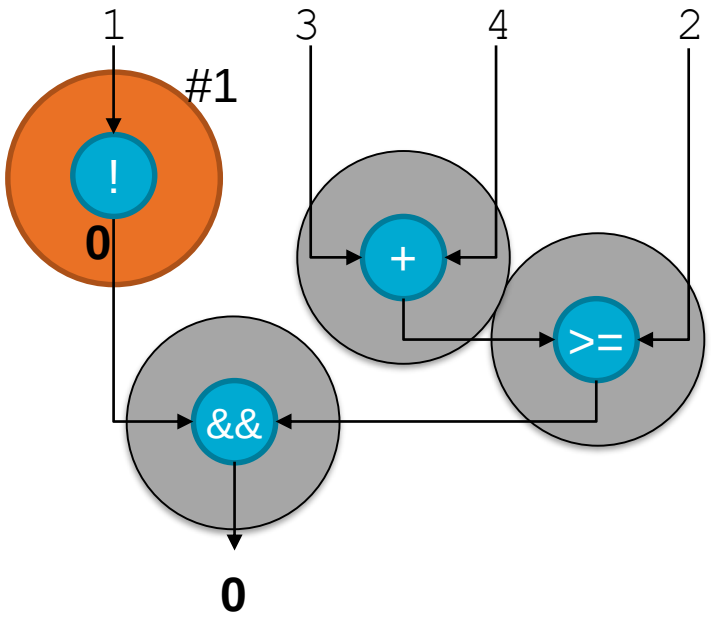
```
int flag=1; int x=3; int y=4; int z=9;

!flag || (x+y >= z)
```



Result: FALSE

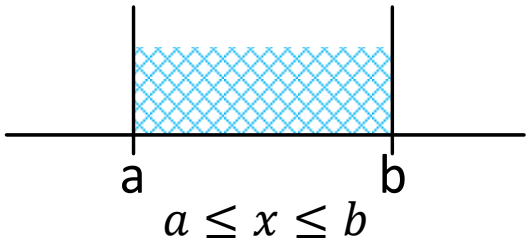
EXAMPLE 3

```
int flag=1; int x=3; int y=4; int z=2;

!flag && (x+y >= z)
```

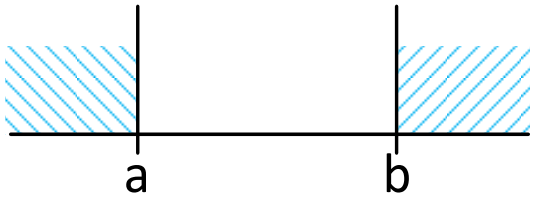


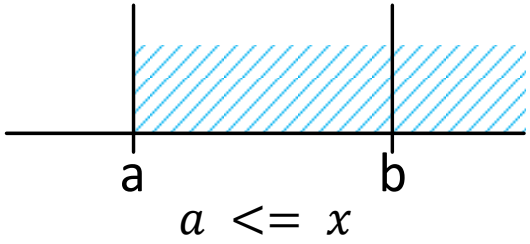
Result: FALSE. By Short-circuit evaluation.

SELECT A RANGE

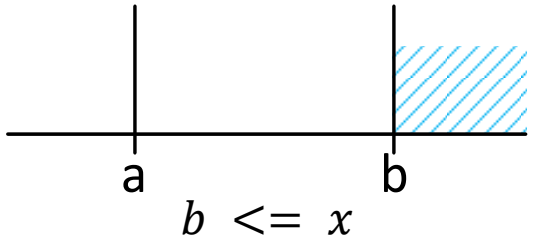


$a \leq x \leq b$

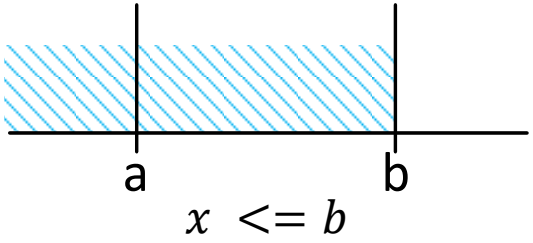




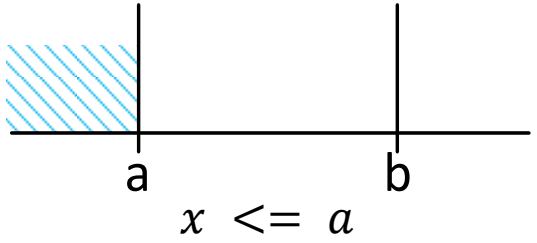
$a <= x$



$b <= x$



$x <= b$



$x <= a$

$a <= x \;\&\&\; x <= b$

$x <= a \;||\; b <= x$

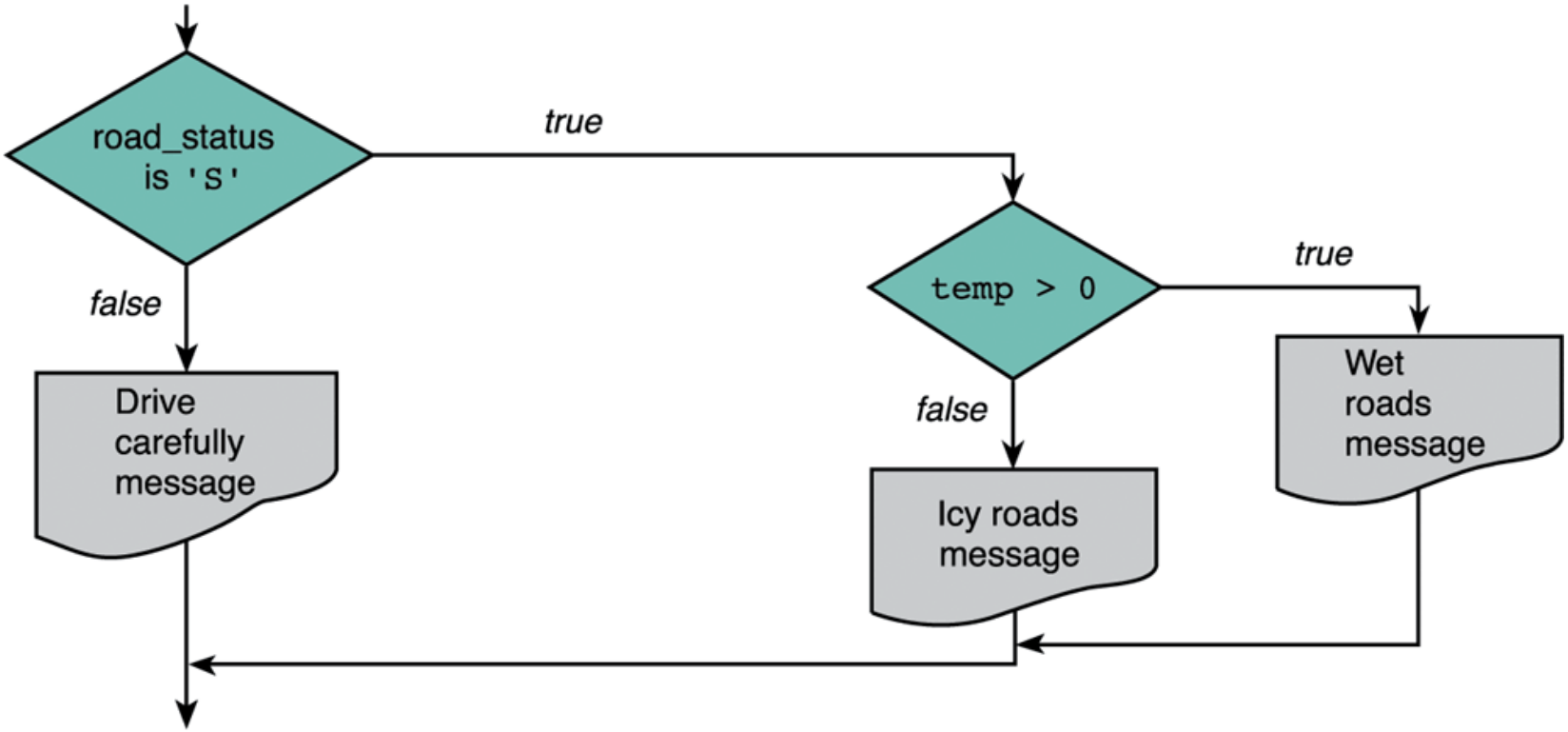
IF STATEMET

Syntax:

```
if (/* logical condition */)
{
  /* block of code to execute if TRUE (1) */
}
else
{
  /* block of code to execute if FALSE (0) */
}
```

You can also write nested if statements.

WRITE A IF STATEMENT BASED ON A FLOWCHART



SWITCH STATEMENT

Looks for an exact match between two values (not recommended for use with float/double).

```
char c='b';
switch (c)
{
  case 'b': printf("b");
            break;
  case 'c': printf("c");
            printf("C");
  case 'd': printf("d");
            break;
  default : printf("X");
}
```

`c='b'` prints `b`, `c='c'` prints `cCd`, `c='e'` prints `X`

Default case is optional.

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

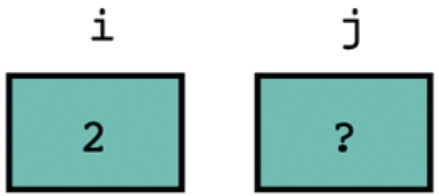
CONTROL STRUCTURES - REPETITION

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

PREFIX AND POSTFIX INCREMENTS

Before...



Increments...

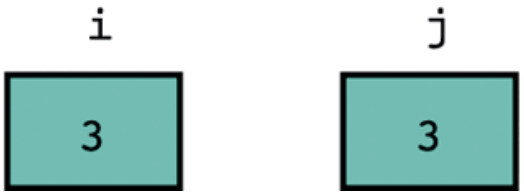
```
j = ++i;
```

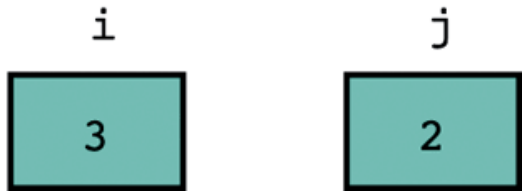
prefix:
Increment `i` and then use it.

```
j = i++;
```

postfix:
Use `i` and then increment it.

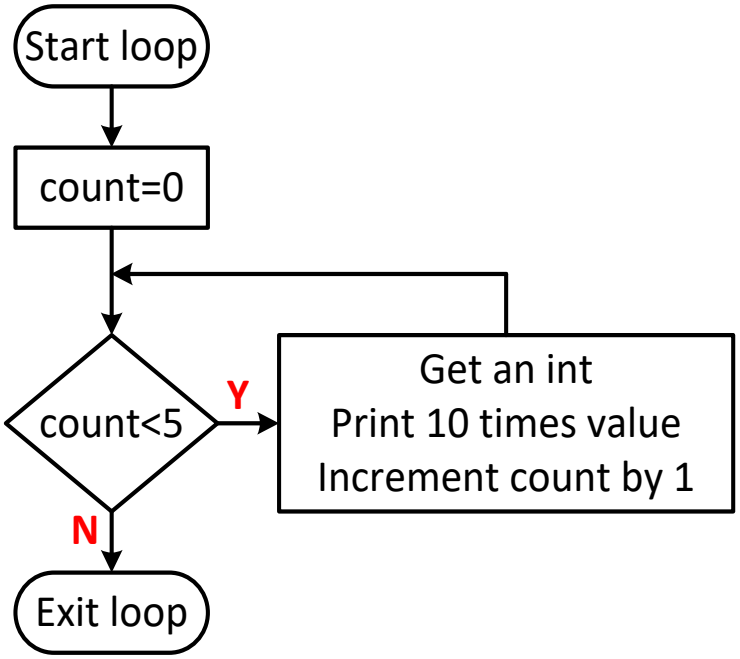
After...





EXAMPLE 1: WHILE LOOP

Collect the user input 5 times and print value*10 on the screen.




```
/* in main */
int count=0;
int inp;
while(count<5)
{
   scanf("%d", &inp);
   printf("%d\n", 10*inp);
   count++;
}
```

Counter controlled while loop

EXAMPLE 1: FOR LOOP

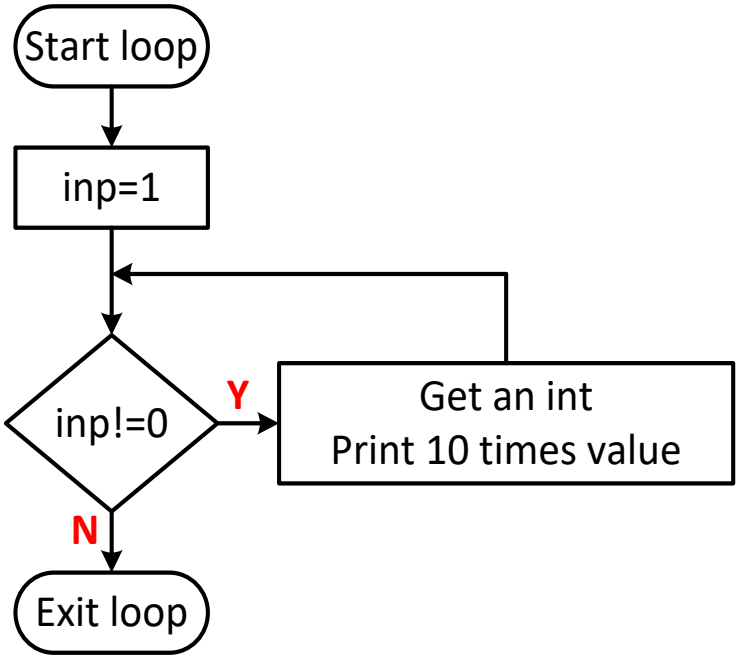
Collect the user input 5 times and print value*10 on the screen.

```
/* in main */
int count;
int inp;
for(count=0; count<5; count++)
{
  scanf("%d", &inp);
  printf("%d\n", 10*inp);
}
```

Counter controlled for loop

EXAMPLE 2: WHILE LOOP

Collect the user input and print value*10 on the screen, until the input is 0.




```
/* before main */
#define SENTINEL 0

/* in main */
int inp=1; /* not 0 */
while(inp!=SENTINEL)
{
  scanf("%d", &inp);
  printf("%d\n", 10*inp);
}
```

Sentinel controlled while loop

EXAMPLE 2: FOR LOOP

Collect the user input and print value*10 on the screen, until the input is 0.

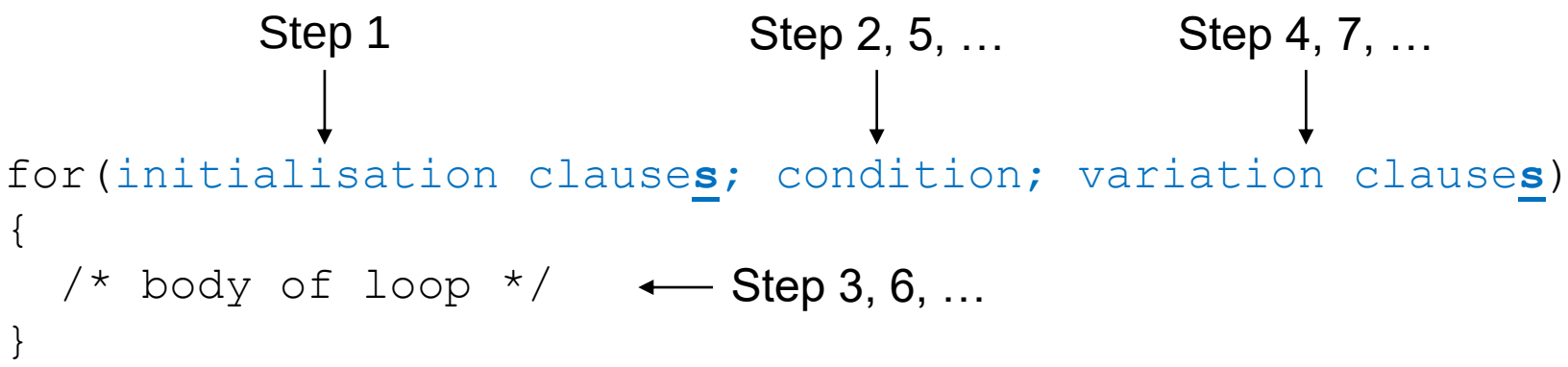
```
/* before main */
#define SENTINEL 0

/* in main */
int inp;
for(inp=1; inp!=SENTINEL; scanf("%d", &inp))
{
  printf("%d\n", 10*inp);
}
```

Sentinel controlled for loop

INITIALISATION, CONDITION AND VARIATION

All loops have three key parts that control the loop – initialisation, condition and variation

Anything that is written as a while loop can be written as a for loop.



For loops are generally preferred in industry over while loops as the loop control statements are better arranged visually.

However when it is not known how many iterations to run while loops "may be" easier.

DO-WHILE LOOP

Do-while loops are used when the body of the loop must always run once prior to making a decision on whether to repeat.

Collect the user input and print value*10 on the screen, until the input is 0. Print the corresponding value for all inputs including 0.

```
/* before main */
#define SENTINEL 0

/* in main */
int inp; /* no need to initialise */
do{
  scanf("%d", &inp);
  printf("%d\n", 10*inp);
} while(inp!=SENTINEL);
```

You can also write nested for, while, do-while loops.