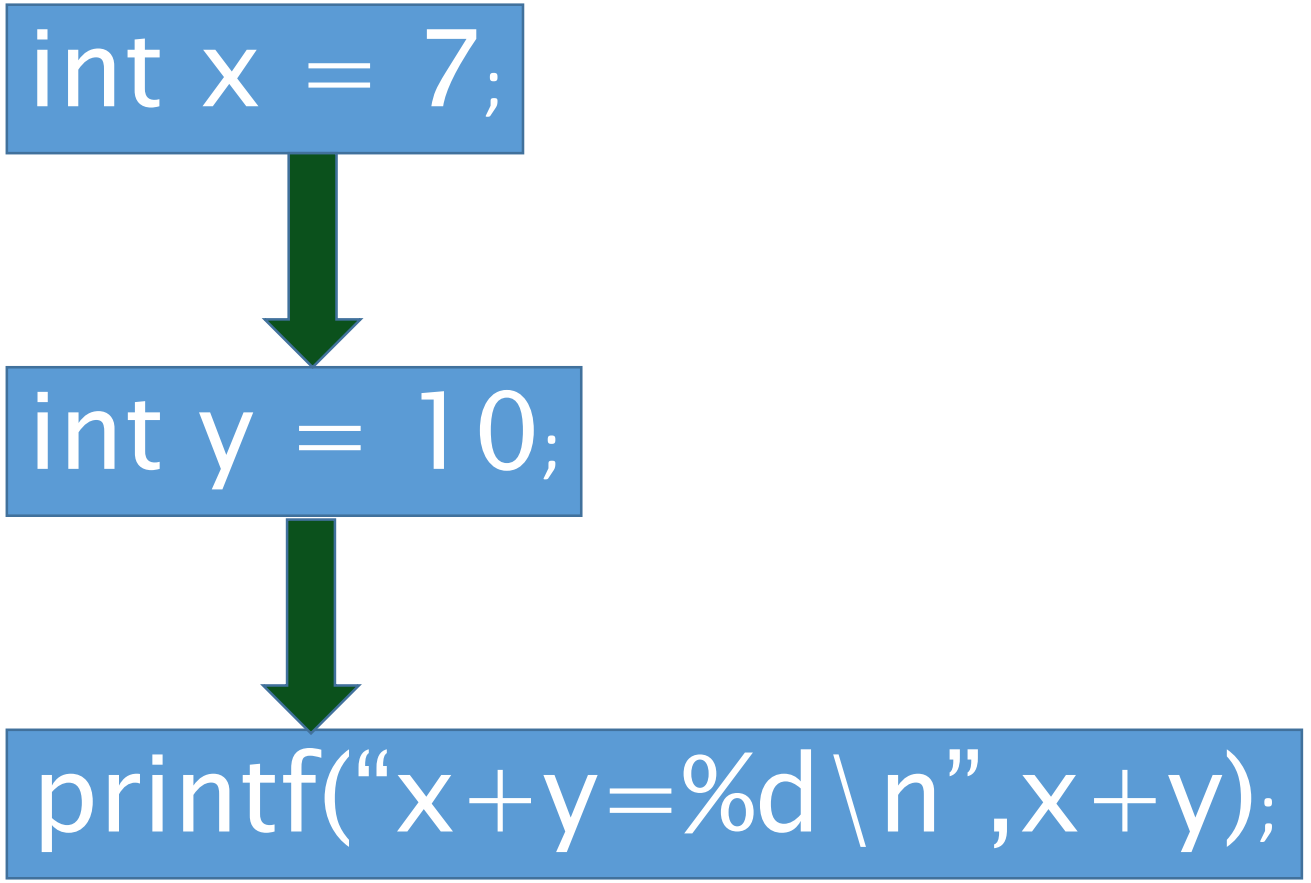
Control Flow

Sequential Control Flow



If

Conditional Processing

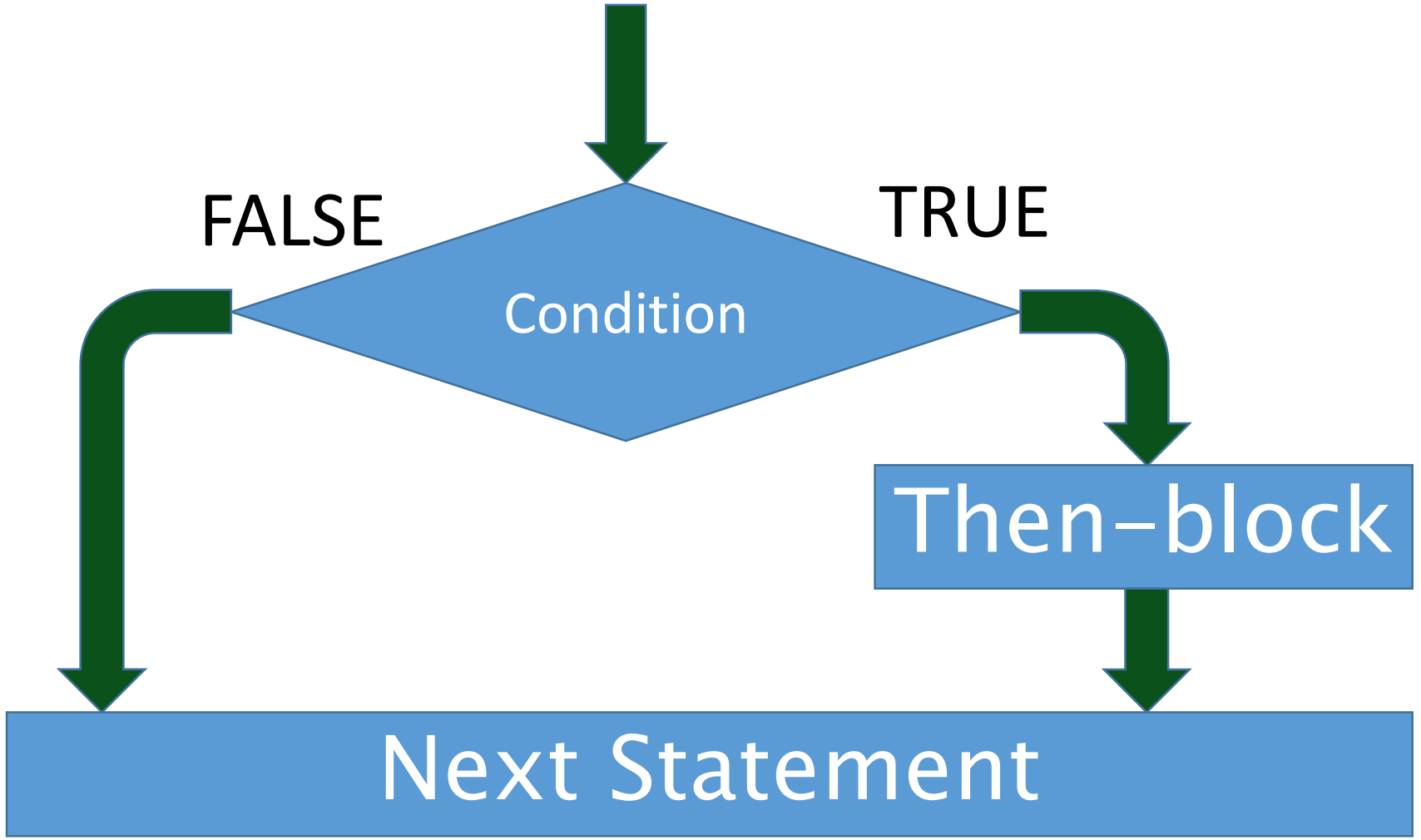
Simple If statement syntax

if (*condition*) *then_statement*;

- *condition* : Any expression whose results is true or false
- *then_statement* : Any statement or block of statements
- The *then_statement* is executed only when the <condition> is true
- Warning: No “then” keyword!

```
if (x!=0) x=113/x;
```

If statement flow



Example If statement

```
if (temp > 35) {
        shut_down();
        printf("Maximum temperature exceeded… shut down\n");
        return ERROR;
}
process_more();
```

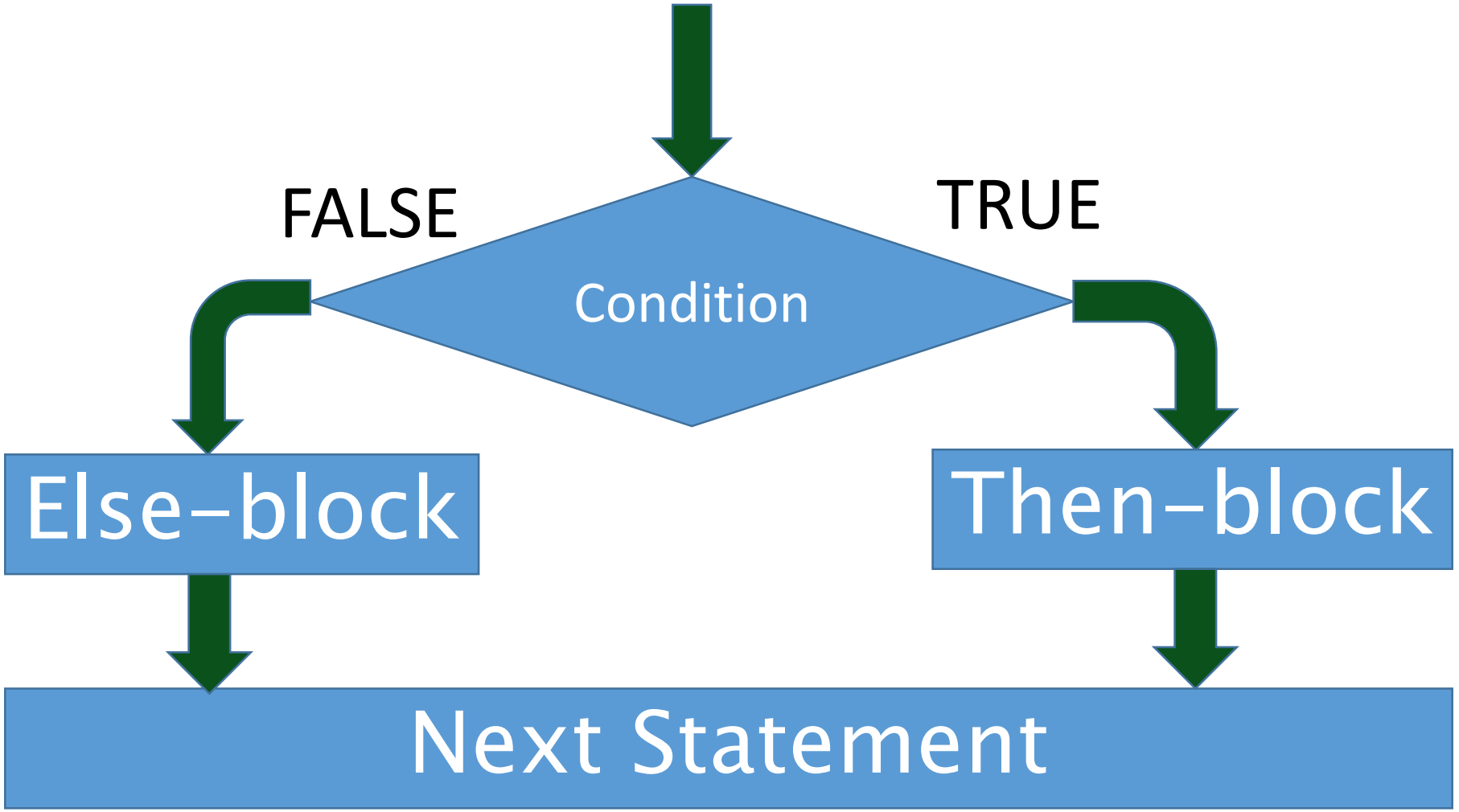
If/Then/Else syntax

if (*condition*) *then_statement* else *else_statement*

- *condition* : Any expression whose results is true or false
- *then_statement* : Any statement or block of statements
- *else_statement* : Any statement or block of statements
- The *then_statement* is executed only when the *condition* is true
- The *else_statement* is executed only when the *condition* is false

```
if (x!=0) x=113/x; else x=-1;
```

If / else statement flow



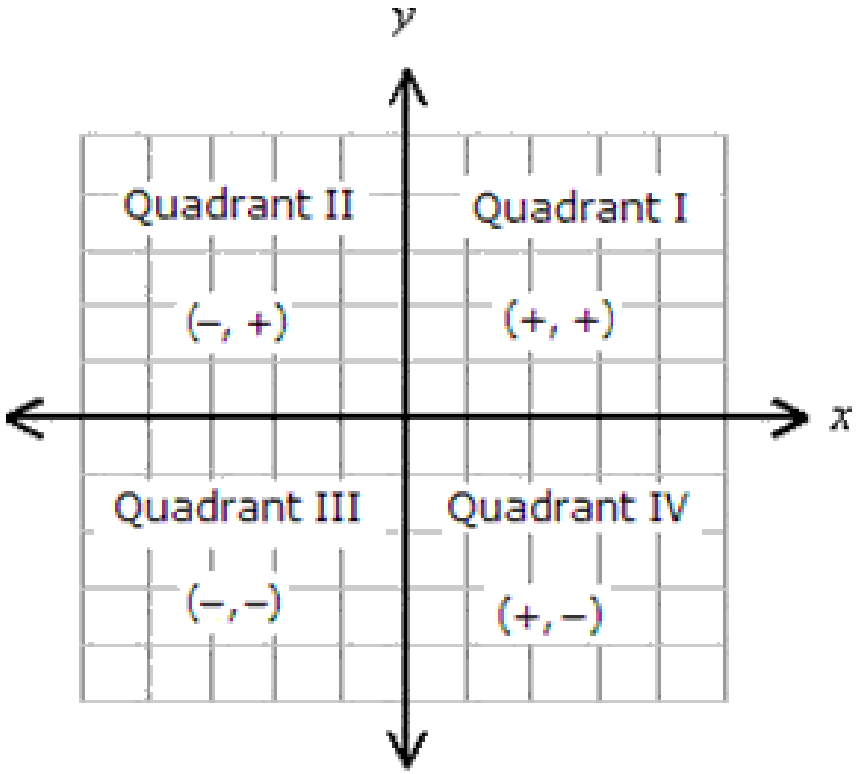
If/Then/Else Example

```
int x = atoi(argv[1]);
if (x<0) {
        printf("Your argument was negative\n");
} else {
        printf("Your argument was positive or zero\n");
}
```

Nested If/Then/Else

- It's perfectly legal for a then block or else block to be an if/then/else statement

```
if (x>=0) {
        if (y>=0) printf("Quadrant I");
        else printf("Quadrant IV");
} else {
        if (y>=0) printf("Quadrant II");
        else printf("Quadrant III");
}
```



Alternative… Compound Logic

```
if ((x>=0)&&(y>=0)) printf (“Quadrant I”);
if ((x>=0)&&(y<0) ) printf(“Quadrant IV”);
if ((x<0) &&(y>=0)) printf(“Quadrant II”);
if ((x<=0)&&(y<0) ) printf(“Quadrant III”);
```

Alternative Else/If Construct

```
if      ((x>=0)&&(y>=0)) printf (“Quadrant I”);
else if ((x>=0)&&(y<0))    printf(“Quadrant IV”);
else if ((x<0)&&(y>=0))    printf(“Quadrant II”);
else                       printf(“Quadrant III”);
```

Deconstructing if/else/if

```
if ((x>=0)&&(y>=0)) { printf ("Quadrant I"); }
else {
     if ((x>=0)&&(y<0)) { printf("Quadrant IV"); }
     else {
          if ((x<0)&&(y>=0)) { printf("Quadrant II"); }
          else {
               printf("Quadrant III");
               }
          }
     }
}
```

Nested If/Then/Else Ambiguity

```
if (x>0) if (y>0) printf (“x>0, y>0\n”);
else printf (“Is x negative, or is y negative????\n”);

if (x>0) { if (y>0) printf (“x>0, y>0\n”);
           else printf(“y<0\n”);
         }

if (x>0) { if (y>0) printf (“x>0, y>0\n”); }
else printf(“x<0\n”);
```

“else” is matched with the nearest “if”

Using "return" instead of "else"

```
if (x<0) z=-1;
else if (y<0) z=-1;
else {
   // x>0 && y>0
   // Calc z from x & y
   …
}
return z;
```

```
if (x<0) return -1;
if (y<0) return -1;
// x>0 && y>0
// Calc z from x & y
…
return z;
```

Don't Forget "?"

```
if (doMore) {
        x=y*3;
} else {
        x=x-1;
}
```

when both *if_statement* and *else_statement* assign values to the same variable

```
x=doMore?(y*3):(x-1);
```

sometimes "?" helps show what is going on

Another else/if construct

comparing the same variable to different literal values

```
if      (op==‘+’) ans=a+b;
else if (op==‘-’)  ans=a-b;
else if (op==‘*’)  ans = a*b;
else if (op==‘/’)  ans = a/b;
else {
        printf “Unrecognized operator: %c\n”,op);
        ans=0;
}
```

Example Switch Statement


```
switch(op) {
        case('+'): ans=a+b; break;
        case('-'): ans=a-b; break;
        case('*'): ans=a*b; break;
        case('/'): ans=a/b; break;
        default: printf("Unrecognized operator: %c\n",op);
                ans=0;
}
```

The "switch" statement

```
switch(expression) {
        case (v1) :
                v1_block
                break;
        case (v2) :
                v2_block
                break;
        …
        default:
                def_block
}
```

```
if (expression==v1) {
        v1_block
} else if (expression==v2) {
        v2_block
}
…
} else {
        def_block
}
```

Loops

Going around in Circles

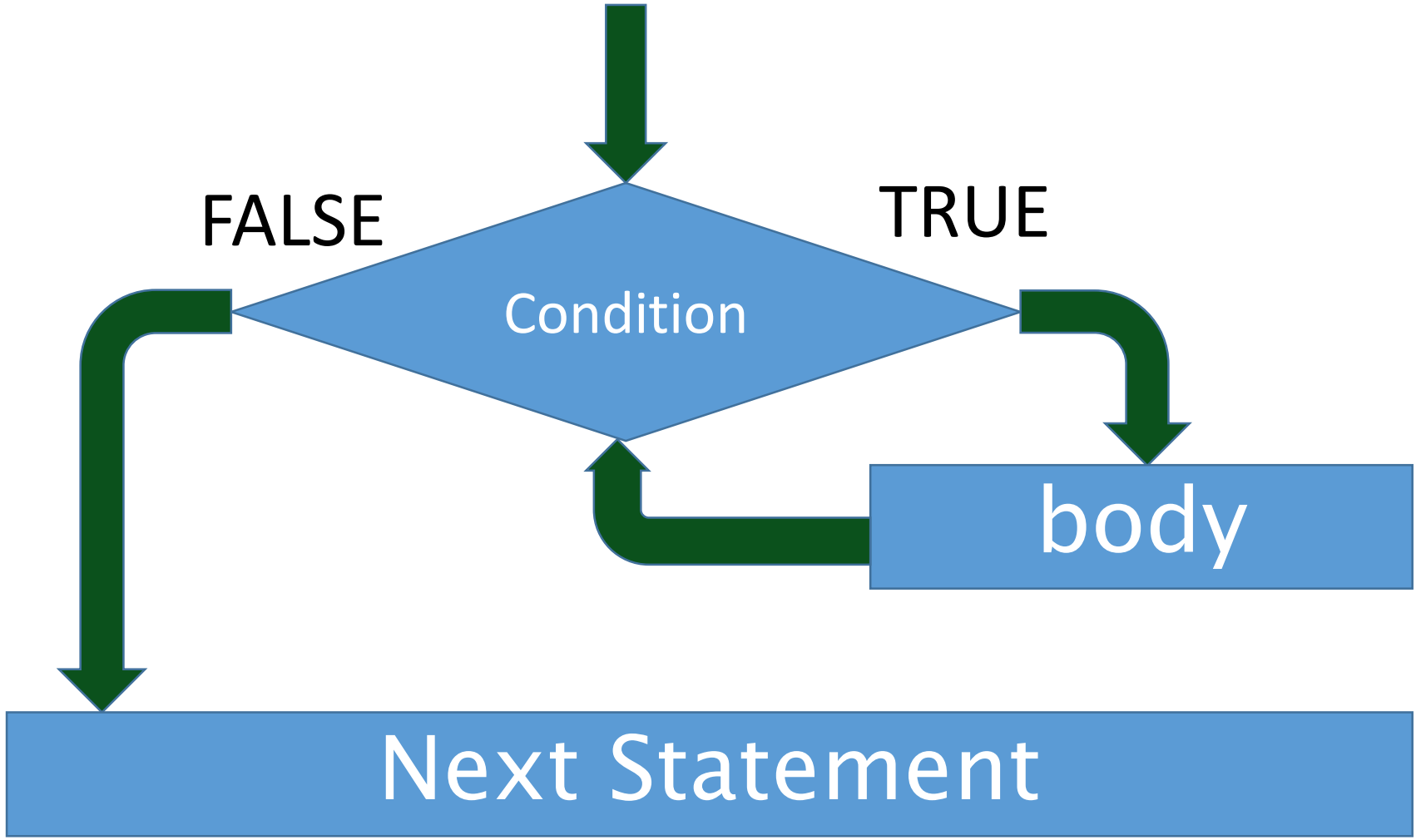
While Loop

while(*condition*) *body*

- *condition* : Any expression evaluated as true or false
- *body*: Any valid statement or block of statements

- *body* is re-executed as long as *condition* is true
- *condition* is re-evaluated after each "iteration" of the loop
- If *condition* is false to start with, *body* is never executed!

while statement flow



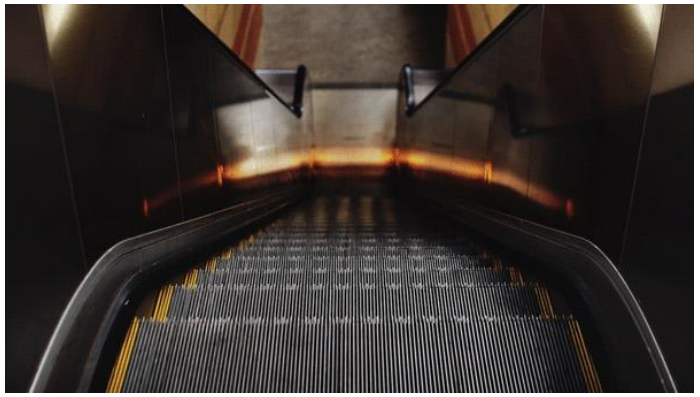
Example While Loop

```
int temp=check_temp();
while(temp<100) {
        add_heat();
        temp=check_temp();
}
/* temperature reached 100… water should boil */
```

Example While Loop

```
int count=100;
while(count>0) {
        int result=experiment(count);
        printf("%3i : ",count);
        int j=0;
        while(j<result) {  printf("*"); j++; }
        printf("\n");
        count--;
}
```

```
100:
 99: *
 98: *
 97: **
 96: *
 95: **
 94: ***
 93: *****
 92: ******
 91: **********
 90: ***********
 89: ********
 88: ******
 87: ****
 86: ***
 85: **
 84: *
...
```



Example While Loop

```
int count=100;
while(count>0) {
        int result=experiment(count);
        printf("%3i : ",count);
        int j=0;
        while(j<result) {  printf("*"); j++; }
        printf("\n");
        count--;
}
```

Initialization

Condition

"Increment"

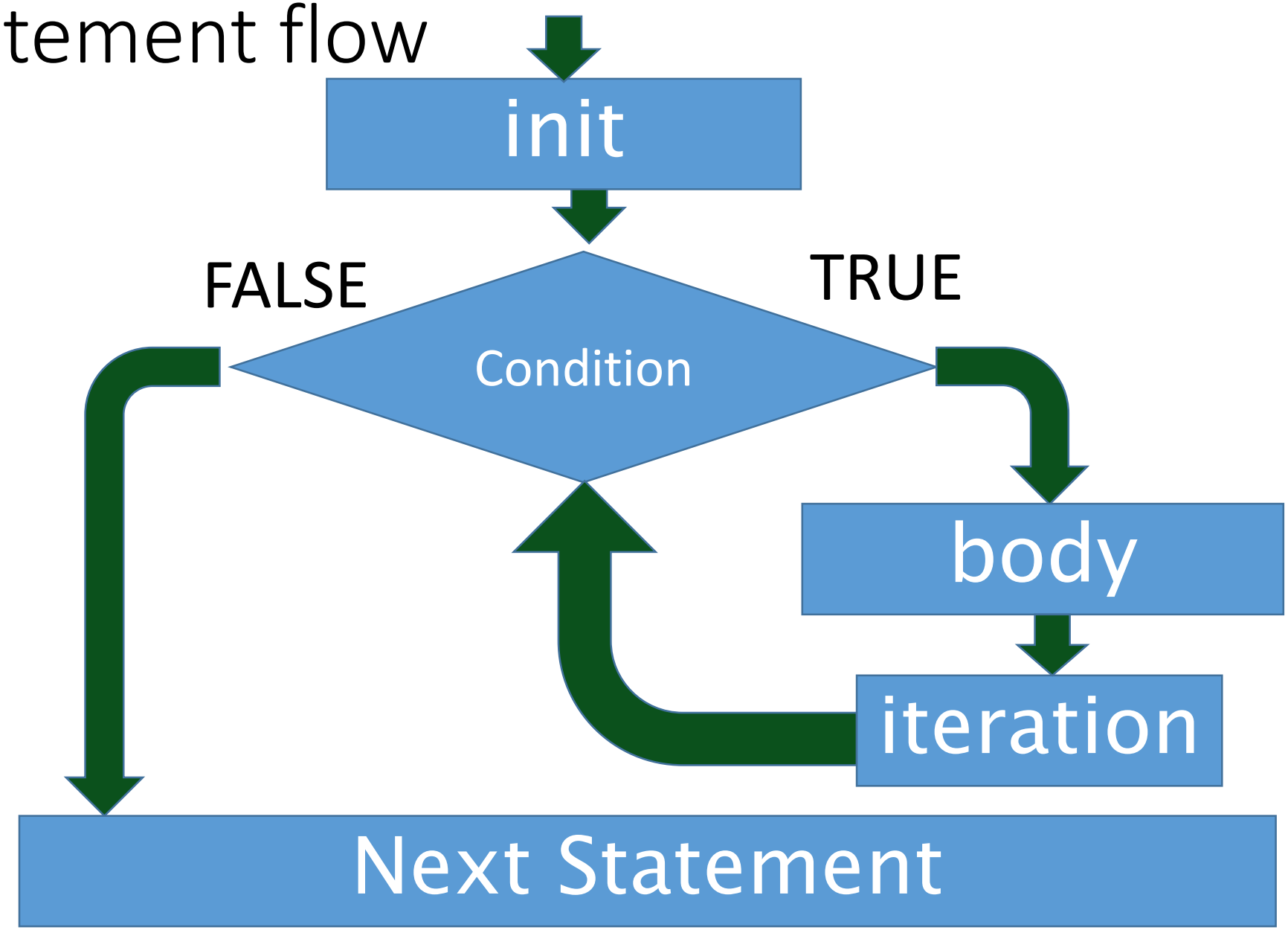
```
100:
 99: *
 98: *
 97: **
 96: *
 95: **
 94: ***
 93: *****
 92: ******
 91: **********
 90: ***********
 89: ********
 88: ******
 87: ****
 86: ***
 85: **
 84: *
...
```

for loop

for (*init*; *condition*; *iteration*) *body*

- *init* : Expression executed before loop starts
- *condition* : Expression evaluated to see if *body* should continue
- *iteration* : Expression executed after *body*, before *condition*
- *body* : Statement or block that makes up the body of the loop

for statement flow



Example for Loop

```
int count;
for(count=100;count>0;count--) {
        int result=experiment(count);
        printf("%3i : ",count);
        int j;
        for(j=0;j<result;j++) printf("*");
        printf("\n");
}
```

```
100:
 99: *
 98: *
 97: **
 96: *
 95: **
 94: ***
 93: *****
 92: ******
 91: **********
 90: ***********
 89: ********
 88: ******
 87: ****
 86: ***
 85: **
 84: *
...
```

More Example For Loops

```
for(data=get_first(); more_data(); data=get_next()) {
        process(data);
}

for(i=0,sum=0; i<100;) sum+=experiment(i++);
average=sum/100;
```

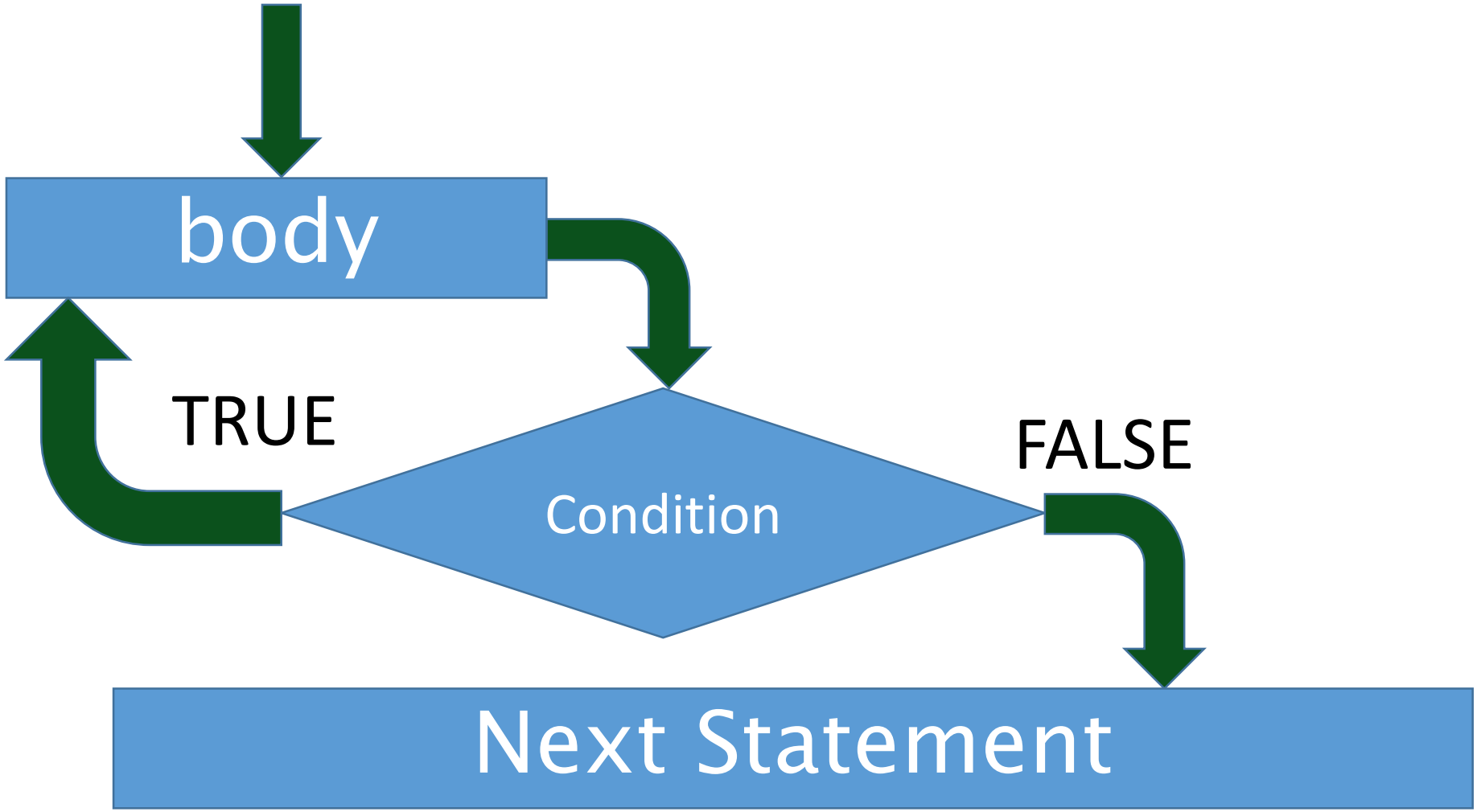
do/while loops

- C also supports a “do while” loop

```
do body while (condition) ;
```

- The only difference from a normal “while” loop is that the body of the loop is executed once BEFORE the condition is tested the first time.
- Very rarely used

do/while statement flow



Infinite Loops (or way too long)

```
int i=get_max();
while(i!=0) {
        process(i);
        i--;
}

for(i=0; i<10; i++) {
        process(i); i=3;
}

x=1;
while(x) { do_stuff(x); }
```

Breaking out of loops early

- break; statement leaves innermost loop (while/for/switch)
  - No checking of <condition>
  - No increment in for loop

```
for(i=0;i<100;i++) {
        result=experiment(i);
        if (result<0) break; /* Something bad happened */
        …
}
```

Early Iteration of Loops

- continue; statement ends *this* iteration of the loop
  - In for loops, iteration statement executed again
  - condition re-evaluated
  - If condition is true, next iteration of the loop starts

```
for(count=0; count<100; count++) {
        if (0==count%7) continue; // skip every 7th experiment
        result=experiment(count);
        …
}
```

Resources

- Programming in C, Chapter 5
- Wikipedia: Conditional (computer programming) (https://en.wikipedia.org/wiki/Conditional_(computer_programming))
- Wikipedia: Control Flow (https://en.wikipedia.org/wiki/Control_flow)