Lecture 2
Control Structures

TIC1001 Introduction to Computing and Programming I

20 Aug 2020

Reminder

Lab starts this Saturday

10:30am to 12:30pm

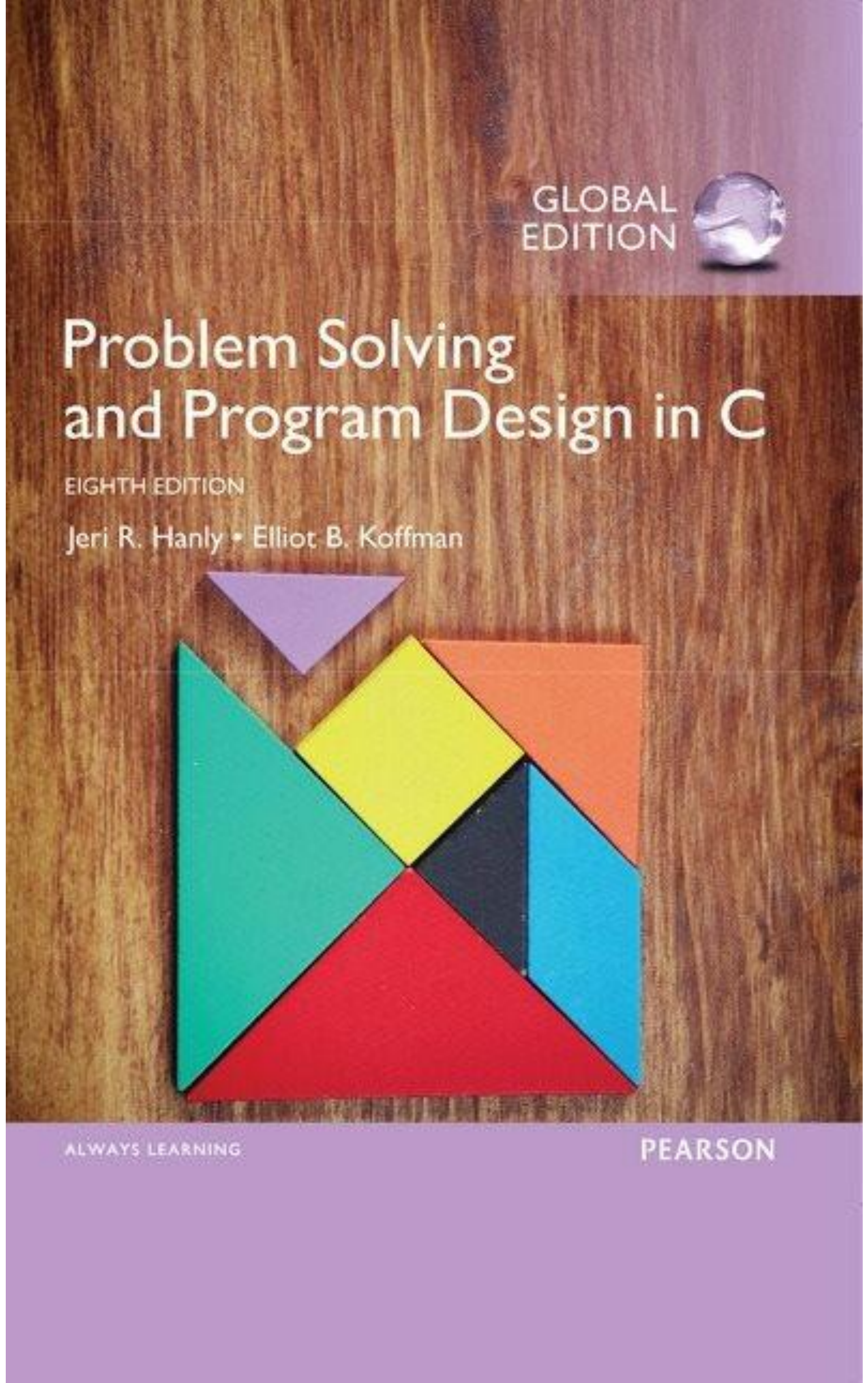
3pm to 5pm

Textbook

Problem solving and program design in C

- Authours: Jeri R. Hanly, Elliot B. Koffman
- ISBN: 9781292098814

- Not compulsory


GLOBAL EDITION
Problem Solving and Program Design in C
EIGHTH EDITION
Jeri R. Hanly • Elliot B. Koffman
ALWAYS LEARNING
PEARSON

Incorrect practice

```
float f_to_c(int degree_f) {
    ...

    float f_to_c = ans;
    return f_to_c;
}
```

This is not Visual Basic .NET

Displaying on Screen

C

```c
#include <stdio.h>

int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

C++

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!" << endl;
    return 0;
}
```

Displaying Numbers

C

```c
int i = 42;
printf("Value of i is: %d\n", i);

int a = 1, b = 2, c = 3;
printf("a: %d, b: %d, c: %d\n", a, b, c);

double d = 25.5;
printf("Value of d is: %f\n", d);
```

Pros
+ formatting

Cons
- Not type safe

C++

```cpp
int i = 42;
cout << "Value of i is: " << i << endl;

int a = 1, b = 2, c = 3;
cout << "a: " << a << ", b: "
     << b << ", c: " << c << endl;

double d = 25.5;
cout << "Value of d is: " << d << endl;
```

Pros
+ type safe

Cons
- verbose

C Escape Sequence

String literal cannot span multiple lines

```
printf("Hello
         World!");
```

Use \n to indicate new line

```
printf("Hello\nWorld!");
```

C++

Adjacent string literals will be concatenated

```
char *poem =
 "No one can tell me,"
 "Nobody knows,"
 "Where the wind comes from,"
 "Where the wind goes."

cout << poem;
```

output

```
No one can tell me,Nobody knows,Where the wind comes from,W
here the wind goes."
```

Escape Sequence

`\n`	Newline (line feed)
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab
`\\`	Backslash
`\'`	Single quote
`\"`	Double quote
`\?`	Question mark (?? can have special meaning)

`printf` format string

`%d` , `%i`	int type as signed decimal number
`%f` , `%F`	double type in normal notation
`%x` , `%X`	unsigned int as hexadecimal number
`%c`	char
`%%`	Prints a literal % character

`printf` format string

- Width field
  - `%5d` prepend with spaces to make 5 characters wide

    `printf("%5d", 123)` will print `  123`
  - `%05d` prepend with 0 to make 5 characters wide

    `printf("%5d", 123)` will print `00123`

- Precision field
  - %.2f Rounds to 2 decimal places

    `printf("%.2f", 3.14156)` will print `3.14`

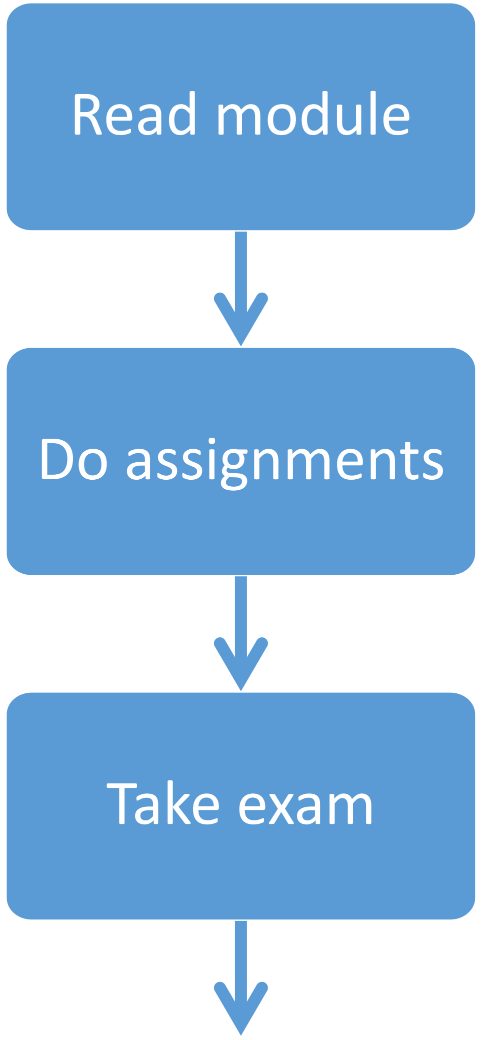
Recall: Elements of Programming

1. Abstraction of the state
   - through variables
   - program
2. Means of mutating state
3. Controlling flow with logic

Control Flow

Control Flow – Sequential

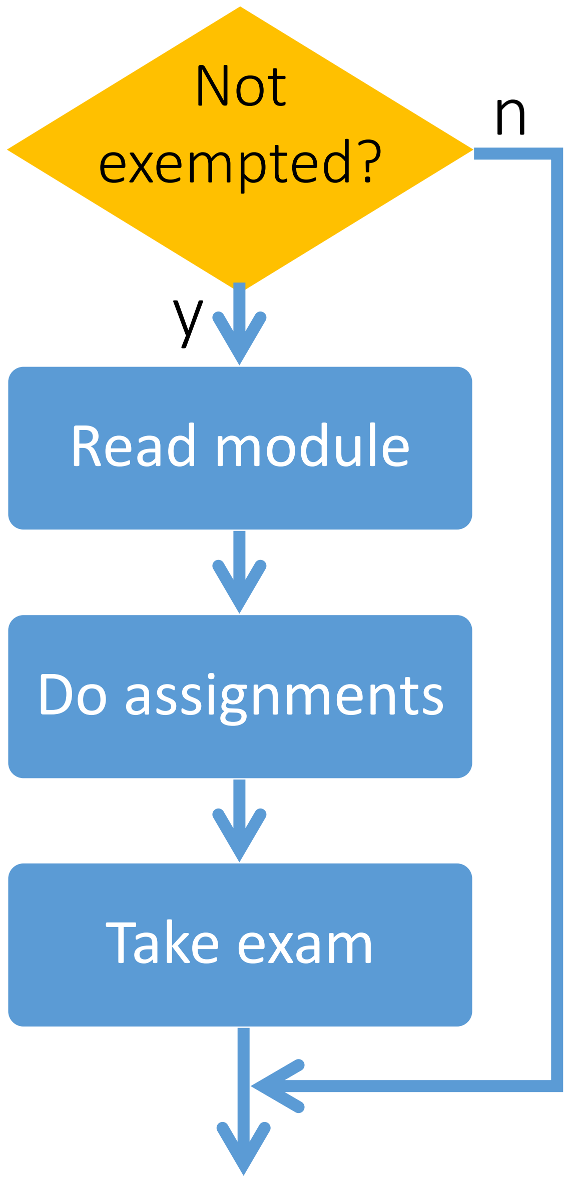
One after another



Control Flow – Conditional

Branch based on a condition

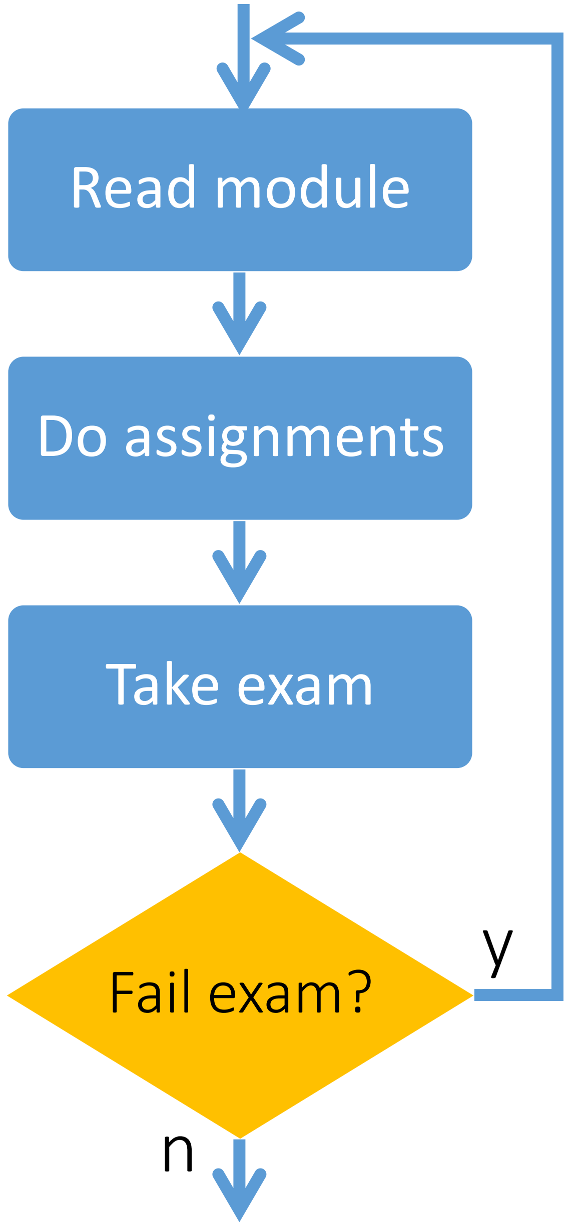
- One sequence of instructions if condition is true
- Another sequence if false (optional)



Control Flow – Loop (Iteration)

Repeat based on a condition

- Keep repeating a sequence of instructions
- As long as the condition is true



How to represent a condition?

Conditional Expressions

Relational Operators:

```
> >= < <= == !=
```

Examples:

- `a > 5`
- `i <= j`
- `5 == 5`
- `x != 0`
- `6 == 6.0`

What is the type of these expressions?

Boolean Type

Name of type

- `bool`

Only two possible values

- `true`
- `false`

Specify `#include <stdbool.h>` to use

- Only C99 and above
- Built-in for C++

Boolean Assignment

Boolean variables

- `bool overweight = true;`
- `bool overweight = (mass/(ht*ht)) > 60;`

Quirk:

- `true` evaluates to integer `1`
- `false` evaluates to integer `0`

Implicit conversion

- `0` to `false`
- anything else to `true`

Logical Operators

`&& (and)   || (or)   ! (not)`

`A && B`

- true if and only if both A and B are true

`A || B`

- true when A or B is true, i.e. between A and B, at least one is true

`!A`

- true if A is false, false if A is true

Truth Tables

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

not	True	False
	False	True

Examples

What's an acceptable weight?

- `(bmi >= 18.5) and (bmi <= 24.9)` ? confusing
- `(18.5 <= bmi) and (bmi <= 24.9)` ✓ acceptable
- `(18.5 <= bmi <= 24.9)` ✗ incorrect

Short-Circuit

C/C++ is lazy. It only does just enough work.

- Only evaluate as much as necessary

`A and B`

- If A evaluates to false, then `A and B` is false.
- So there is no need to evaluate B.

`A or B`

- If A evaluates to true, then `A or B` is true.
- So there is no need to evaluate B.

Short Circuit - Examples

```
(bmi > 24.5) and is_smoker(person)
```

- If bmi is <= 24.5, the first part is false
- the function is_smoker is never called

```
(bmi < 24.5) or is_strong(person)
```

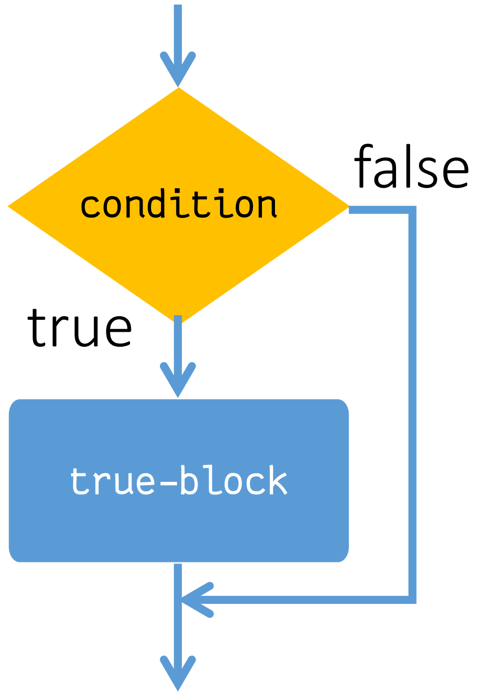
- If bmi is < 24.5, the first part is true
- the function is_strong is never called

Conditional

if Statement

```
if (booleanExpression) {
    ...
    // true-block
    ...
}
```

- Executes `true-block` block when `booleanExpression` is true.



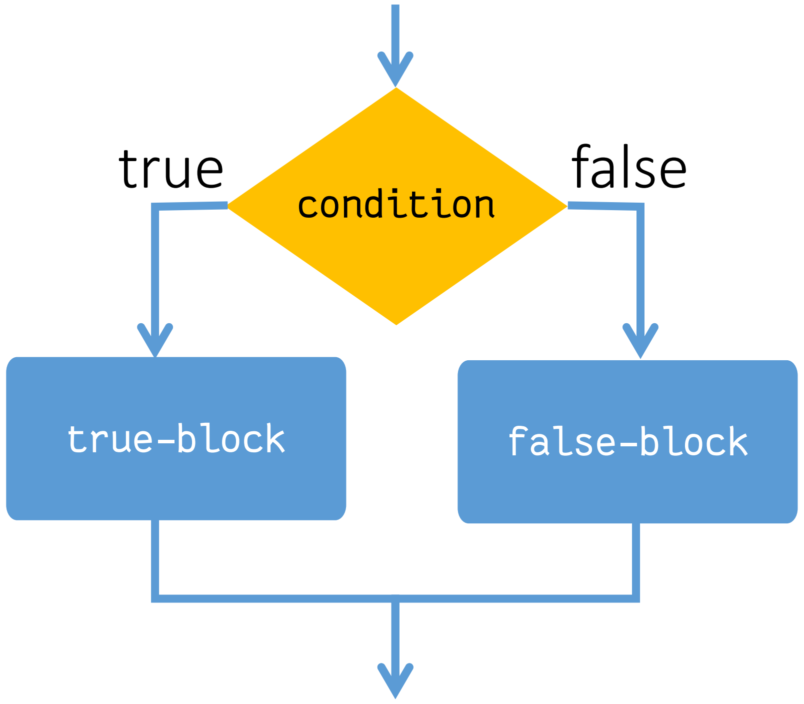
Statement block

```
if (booleanExpression) {
    ...
    // true-block
    ...
}
```

matching braces denotes a block

if-else Statement

```
if (booleanExpression) {
    ...
    // true-block
    ...
} else {
    ...
    // false-block
    ...
}
```



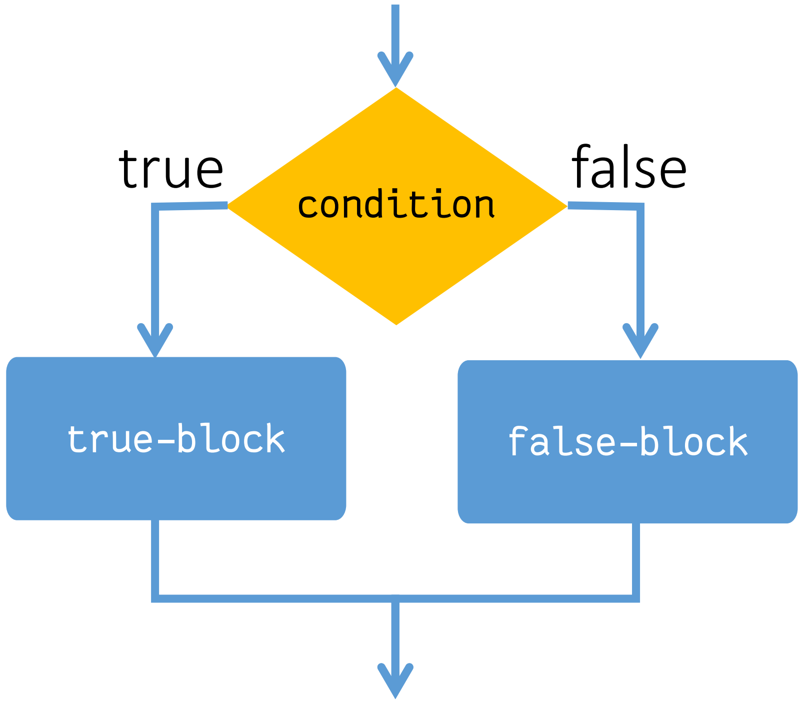
Executes `true-block` block when `booleanExpression` is `true`, otherwise execute `else-block`.

if-else Statement

For block consisting of one statement, curly braces may be omitted.

```
if (x > 0)
    i = i + 1;
else
    i = i - 1;
```

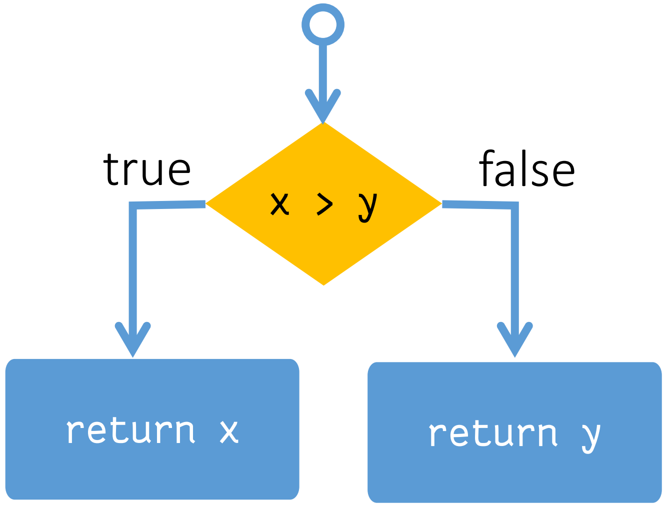
This is strongly discouraged!



Example: Max of two numbers

Using `if..else`

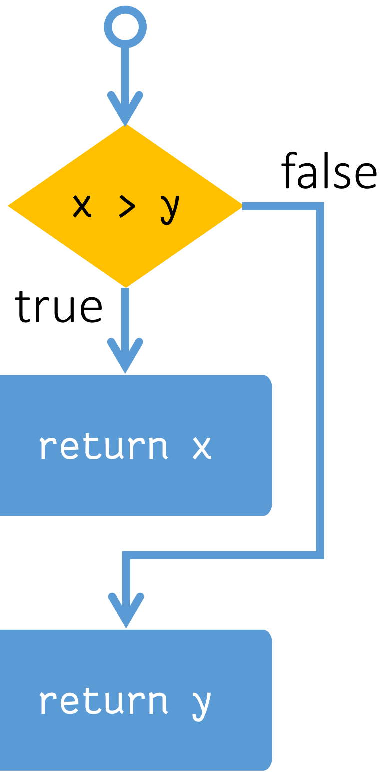
```
int max(int x, int y) {
    if (x > y) {
        return x;
    } else {
        return y;
    }
}
```



Example: Max of two numbers

Using one `if`

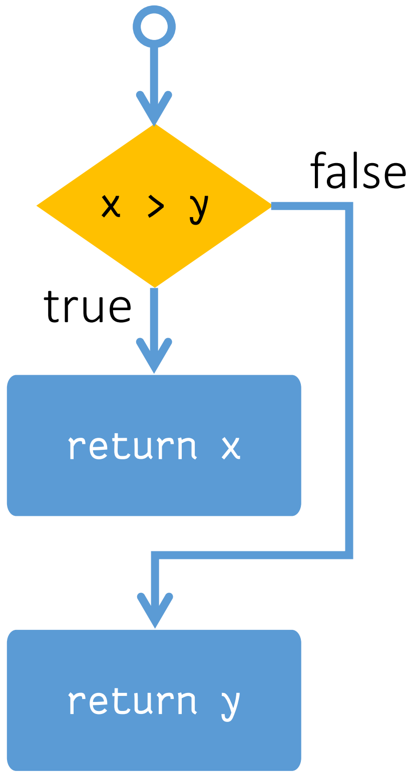
```
int max(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```



Example: Max of two numbers

Suppose we do not return, but just display the number.

```
int max(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

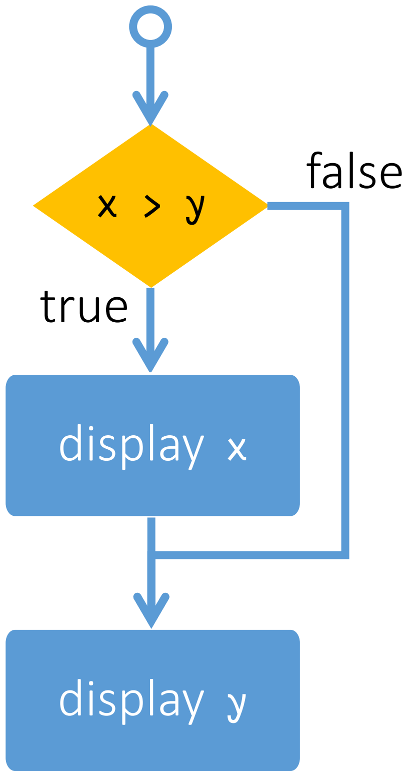


Example: Max of two numbers

Suppose we do not return, but just display the number.

```
void print_max(int x, int y) {
    if (x > y) {
        cout << x << endl;
    }
    cout << y << endl;
}
```

Suppose `x=5, y=3`

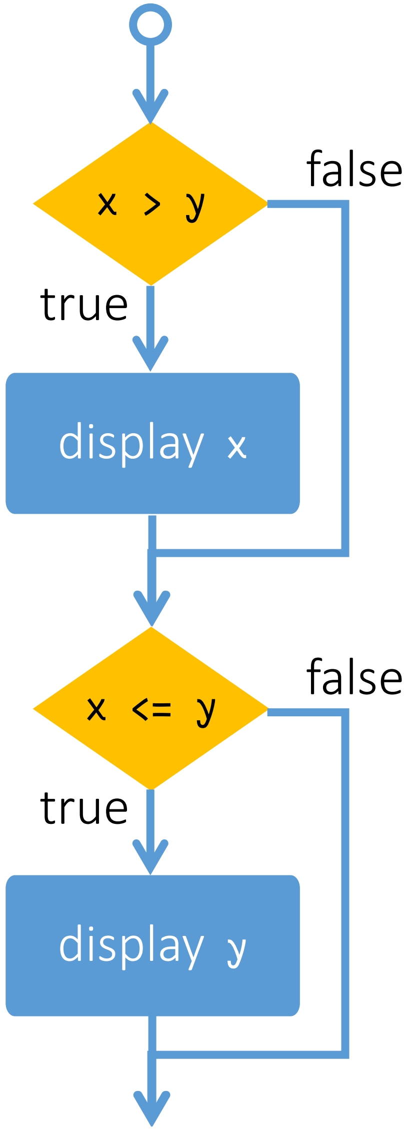


Example: Max of two numbers

Add another `if` statement

```
void print_max(int x, int y) {
    if (x > y) {
        cout << x << endl;
    }
    if (x <= y) {
        cout << y << endl;
    }
}
```

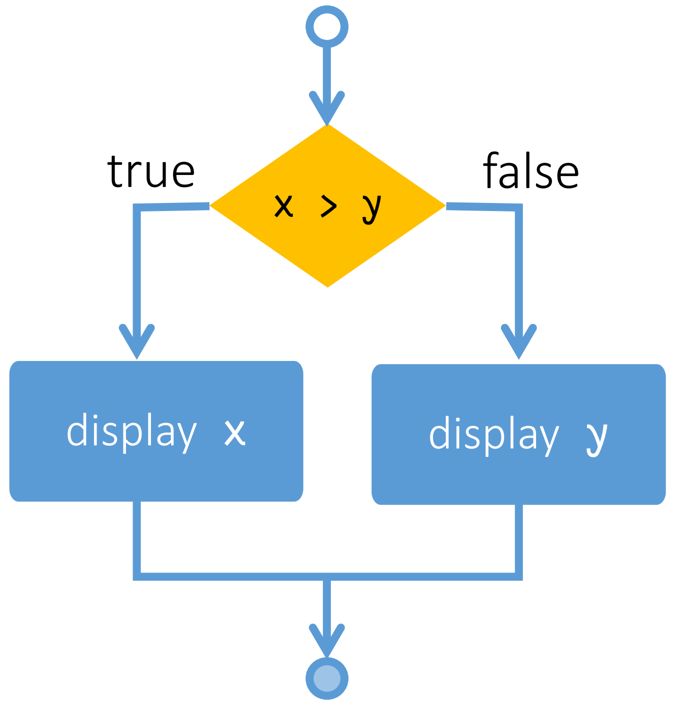
So confusing!



Example: Max of two numbers

Using `if..else`

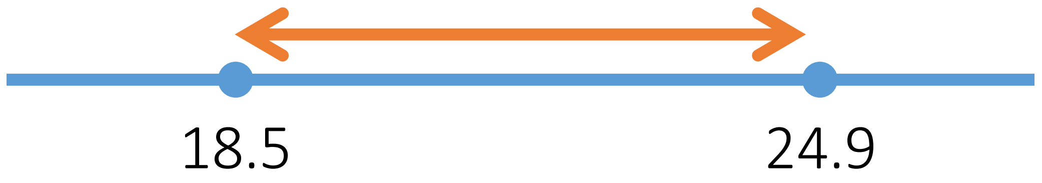
```
void print_max(int x, int y) {
    if (x > y) {
        cout << x << endl;
    } else {
        cout << y << endl;
    }
}
```



Example: BMI

Using logical and

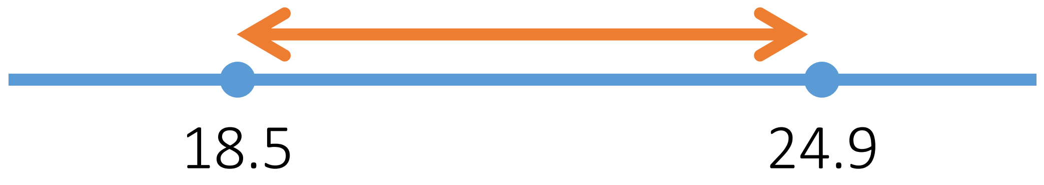
```
bool is_ideal_weight(double bmi) {
    if (bmi >= 18.5 and bmi <= 24.9) {
        return true;
    } else {
        return false;
    }
}
```



Example: BMI

Magic

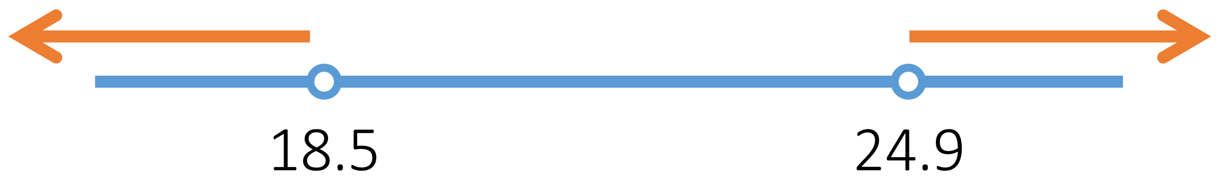
```
bool is_ideal_weight(double bmi) {
    return (bmi >= 18.5 and bmi <= 24.9);
}
```



Example: BMI

Using logical `or`

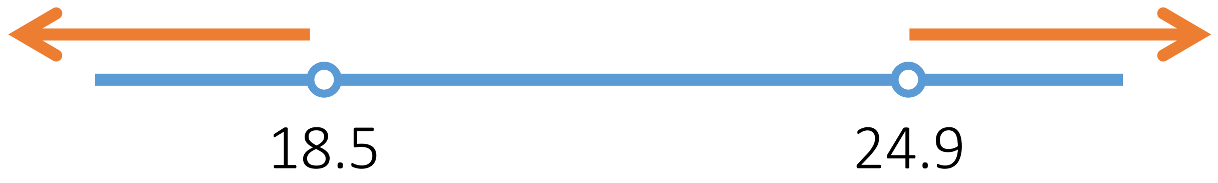
```
bool is_ideal_weight(double bmi) {
    if (bmi < 18.5 or bmi > 24.9) {
        return false;
    } else {
        return true;
    }
}
```



Example: BMI

Magic

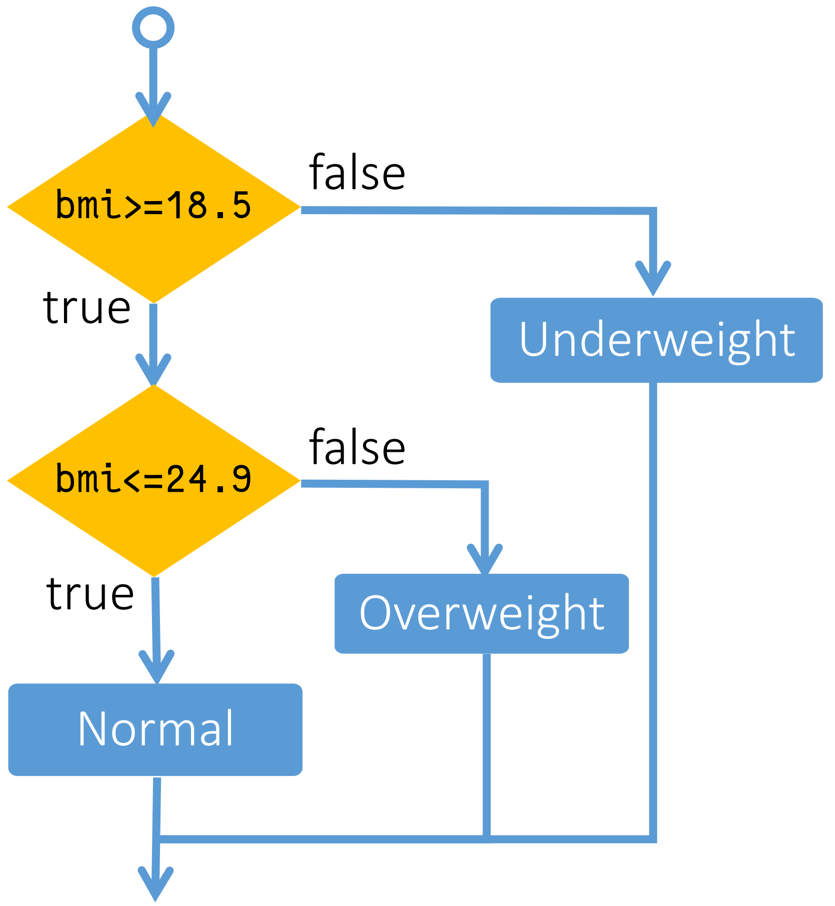
```
bool is_ideal_weight(double bmi) {
    return !(bmi < 18.5 or bmi > 24.9);
}
```



Nested if..else

Three different output

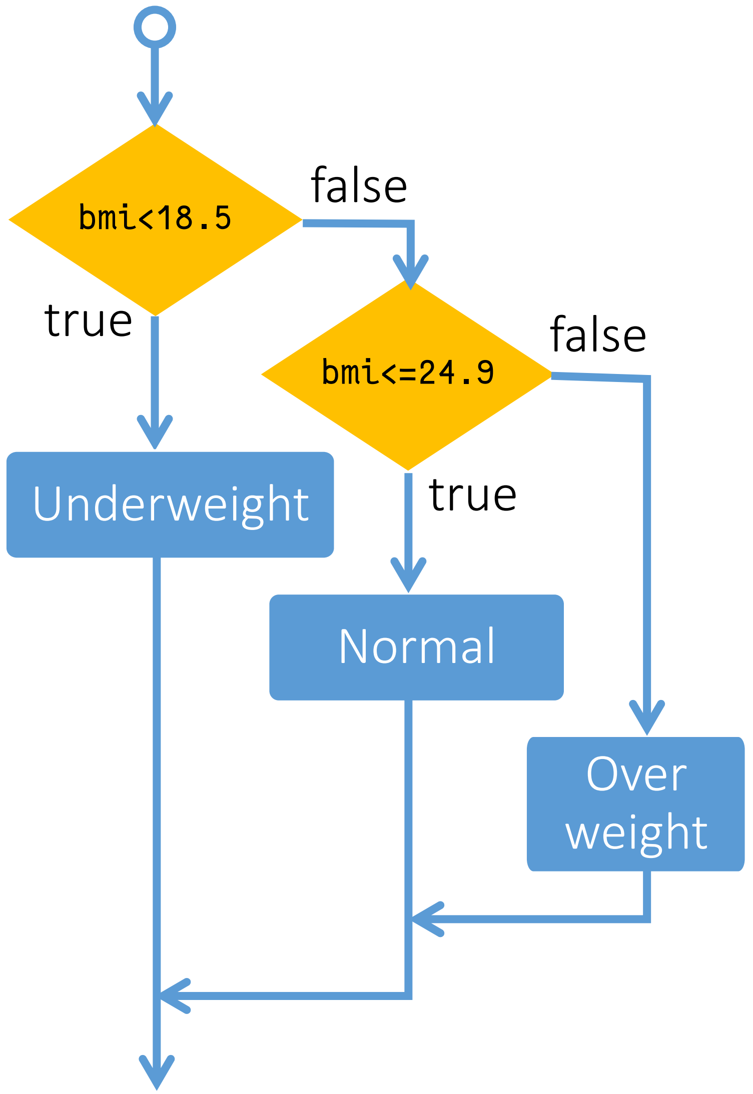
```
void check_bmi(double bmi) {
    if (bmi >= 18.5) {
        if (bmi <= 24.9) {
            cout << "Normal";
        } else {
            cout << "Overweight";
        }
    } else {
        cout << "Underweight";
    }
}
```



Nested if..else

Three different output

```
void check_bmi(double bmi) {
    if (bmi < 18.5) {
        cout << "Underweight";
    } else {
        if (bmi <= 24.9) {
            cout << "Normal";
        } else {
            cout << "Overweight";
        }
    }
}
```

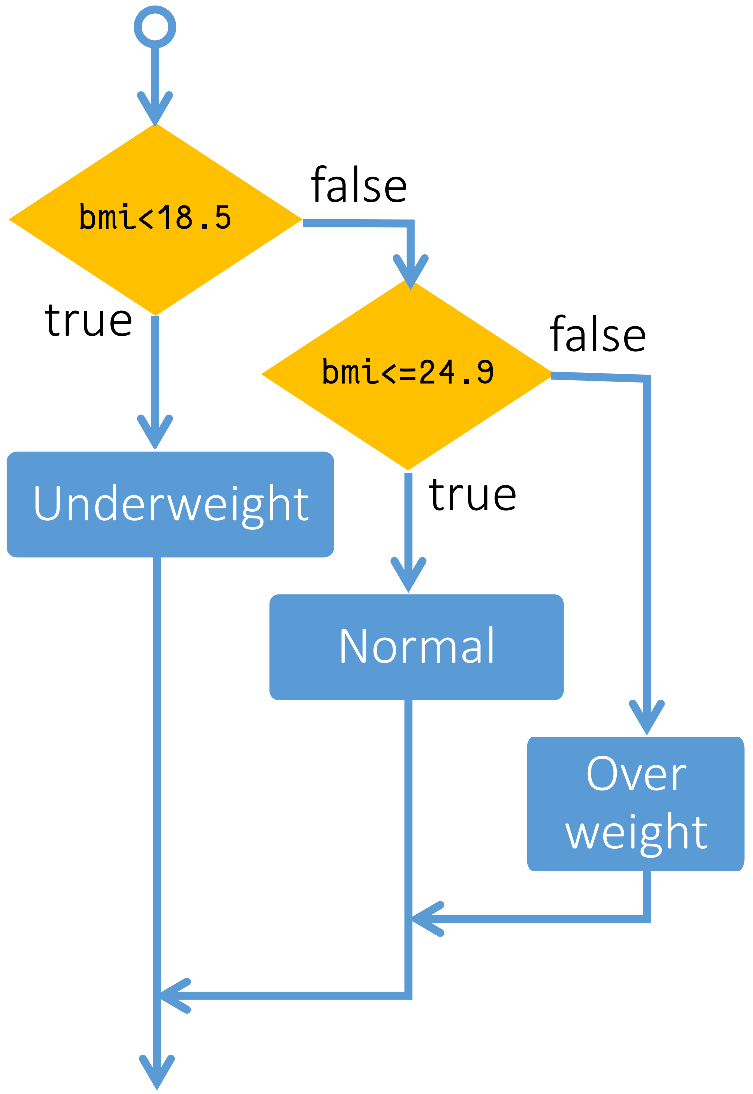


Nested if..else

Three different output

```
void check_bmi(double bmi) {
    if (bmi < 18.5) {
        cout << "Underweight";
    } else if (bmi <= 24.9) {
        cout << "Normal";
    } else {
        cout << "Overweight";
    }
}
```

Removing the redundant braces makes it more readable.



Looping/Iteration

Looping/Iteration Statement

Repeat statements in a block as long as a condition is true

Three forms

- `do..while` loop
- `while` loop
- `for` loop

Condition must eventually become false

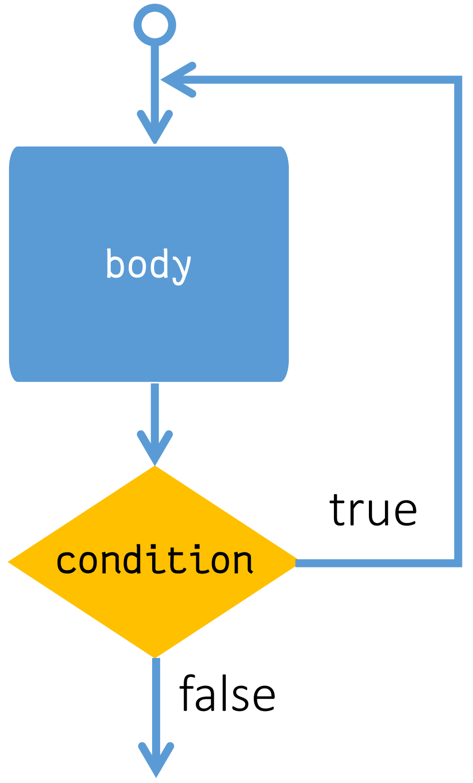
- otherwise loop will never terminate (infinite/endless loop)

do..while Statement

```
do {
    // body
} while (condition);
```

body is always executed at least once

Note the semicolon at the end

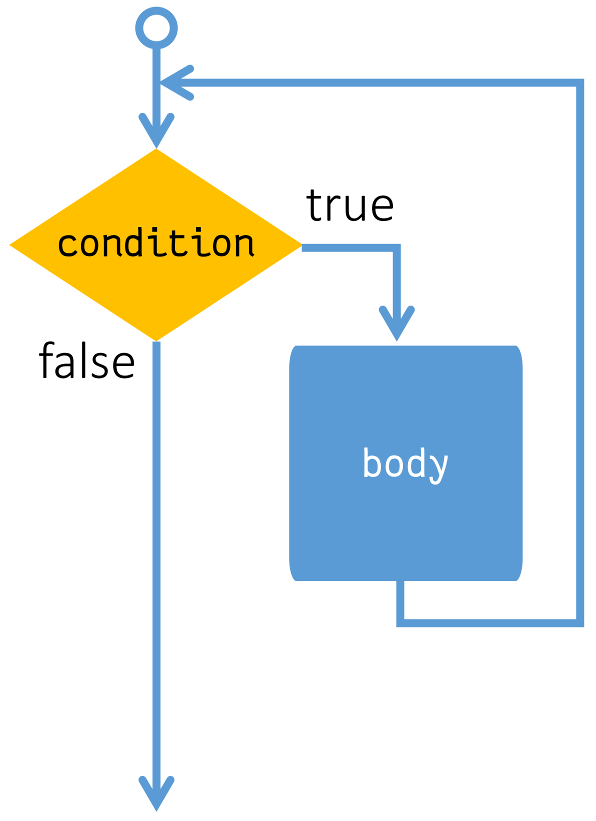


Example: Sum 1 to 100

```
// Sum from 1 to 100
int sum = 0, number = 1;
do {
    sum += number;
    number += 1;
} while (number <= 100);
```

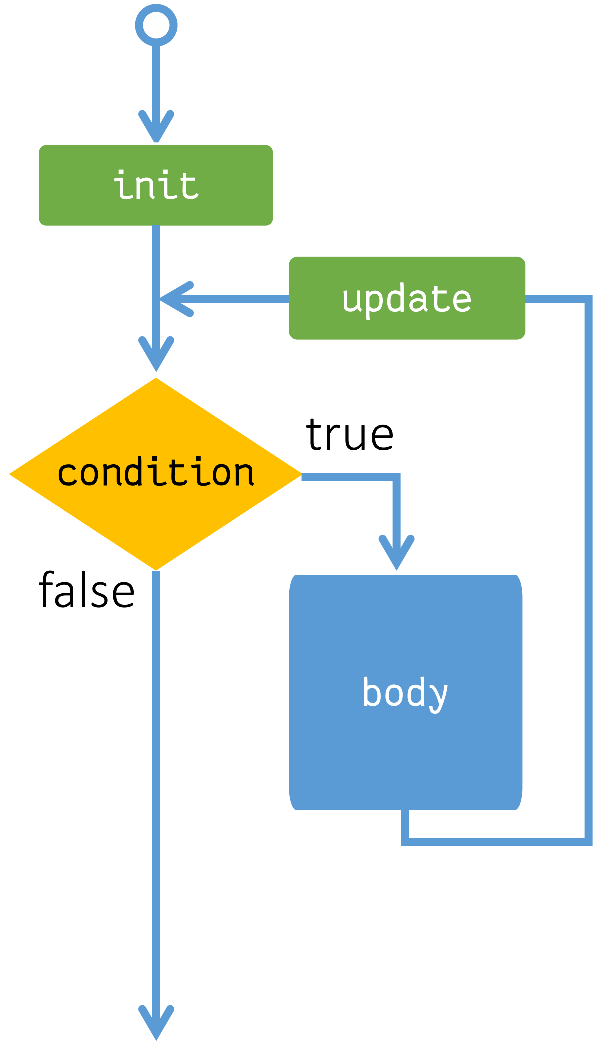
while Statement

```
while (condition) {
    // body
}

// Sum from 1 to 100
int sum = 0, number = 1;
while (number <= 100) {
    sum += number;
    number += 1;
}
```



for Statement

```
for (init; condition; update) {

    // body
}

// Sum from 1 to 100
int sum = 0;
for (int i = 1; i <= 100; ++i) {
    sum += i;
}
```



for loop vs while loop

```
// while loop
int sum = 0, i = 1;
while (i <= 100) {
    sum += i;
    i += 1;
}

// for loop
int sum = 0;
for (int i = 1; i <= 100; ++i) {
    sum += i;
}
```

Infinite loops

```
while (1) {

}
```

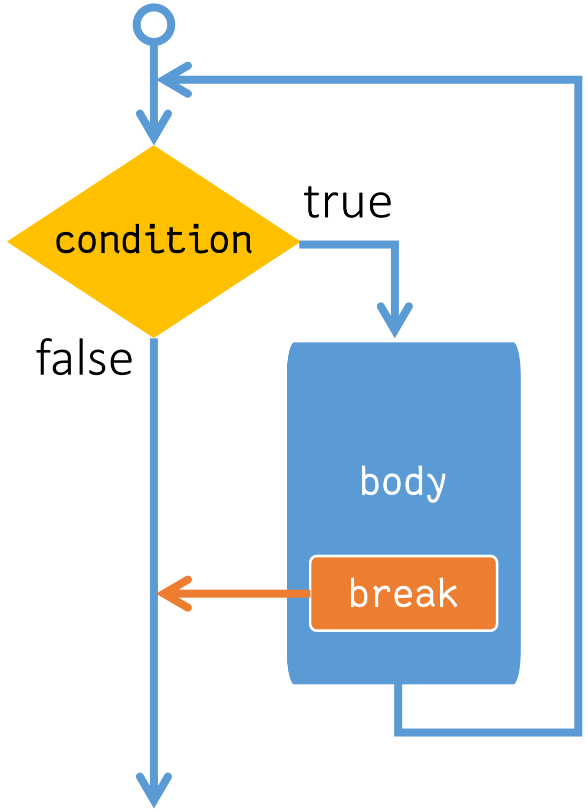
```
for (;;) {

}
```

Interrupting loops

break

- exits the loop body immediately

```
int sum = 0;
for (int i = 1; i <= 100; ++i) {
    if (i > 50) { break; }
    sum += i;
}
```

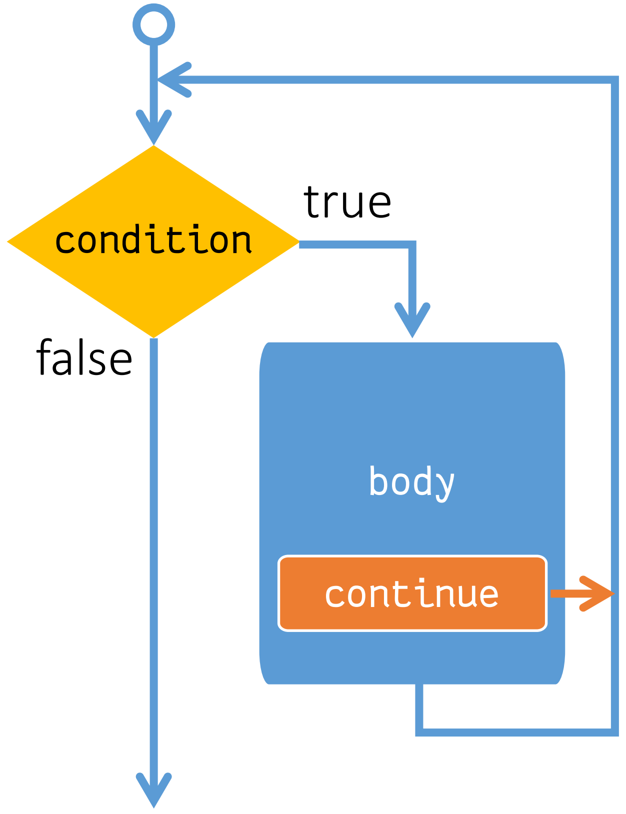


Interrupting loops

continue

– go back to start of loop immediately

```
int sum = 0;
for (int i = 1; i <= 100; ++i) {
    if (i % 2) { continue; }
    sum += i;
}
```



Summary

Conditional (Boolean) Expressions

- `True` or `False` value

Branching

- `if..else` Do this or that
- Nested to make even more conditions

Looping

- `do..while(cond);`
- `while(cond)`
- `for(init; cond; update)`
- `break` and `continue`

Tasks

Lecture Training

- Bonus cut-off by Monday 23:59

Problem Set 2

- Due in two/three weeks?
- PS1 due next Sunday

Tutorial 1

- Discussion the following Sat

Lab session

- This coming Sat
- 10:30am and 3:00pm