We are 183

L07: Week 5 - Monday

Due Soon

- Project 2 – Friday

- Exam 1: Two Weeks!
  - This Wednesday, February 3rd: last day to request alternate time (4pm)

Last Time… on EECS 183

Comparing *string*
Comparing *double*
Nested *if-else* statements

char by char: ASCII followed by length

```
if("apples" == "apples") {
    …
}
```

1. Compare two strings character-by-character
2. Compare using the ASCII value of each character
    1. 0-9 in order
    2. A-Z in order, but case sensitive
3. If all characters match, the longer string is greater

i>Clicker #1

```
string s1 = "Apples123";
string s2 = "apples45";
if(s1 < s2) {
    cout << s1;
}
else {
    cout << s2;
}
```

What does this code snippet print?

A. Apples123
B. apples45
C. Apples123apples45

i>Clicker #1

```
string s1 = "Apples123";
string s2 = "apples45";
if(s1 < s2) {
    cout << s1;
}
else {
    cout << s2;
}
```

What does this code snippet print?

A. Apples123
B. apples45
C. Apples123apples45

i>Clicker #2

```
double ratio = 0.7;
if(pow(ratio, 2) >= 0.49) {
    cout << "Great job";
}
else {
    cout << "Work harder";
}
```

What does this code snippet print?

A. Great job
B. Work harder

i>Clicker #2

```
double ratio = 0.7;
if(pow(ratio, 2) >= 0.49) {
    cout << "Great job";
}
else {
    cout << "Work harder";
}
```

What does this code snippet print?

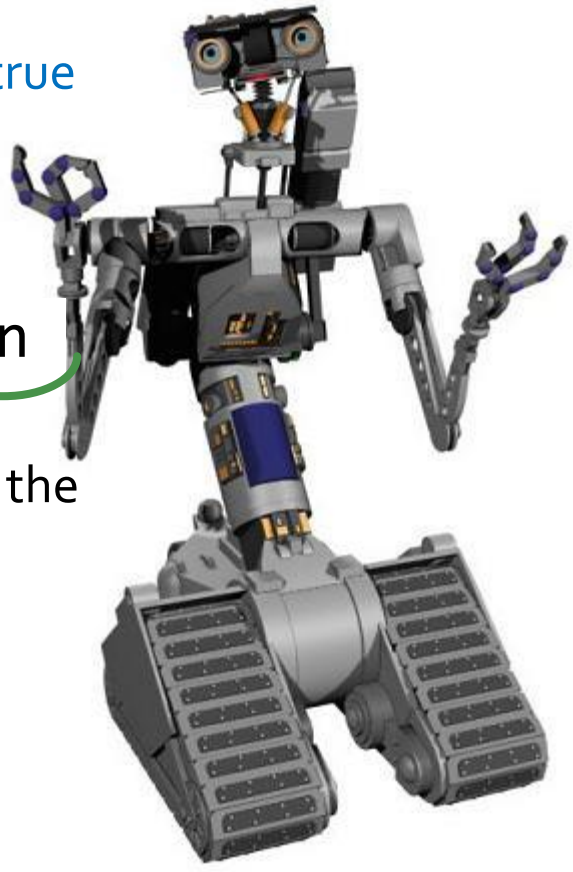
A. Great job

B. Work harder

Short-circuit evaluation

```
condition && condition && condition
```

Expression evals to true if and only if **ALL** conditions are true



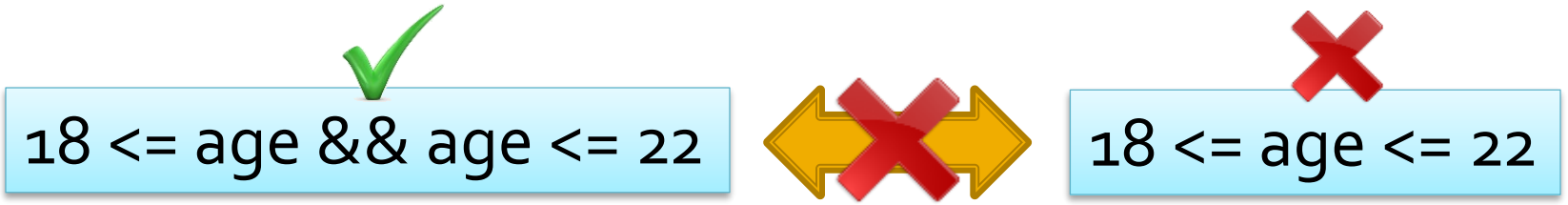
```
condition || condition || condition
```

Expression evals to true if and only if **AT LEAST ONE** of the conditions is true

```
!condition
```

Expression evals to true if and only if *condition* is false

Short-circuit evaluation




```
(i == 5)  &&  (j > 10)
```

If the left operand of && is false

Right operand is not evaluated

```
(i == 5)  ||  (j > 10)
```

If the left operand of || is true

Right operand is not evaluated

i>Clicker #3

```
if(foo(count1) < 5 && foo(count2) < 49) {
    cout << "Within range";
}
else {
    cout << "Out of range";
}
```

How many times will foo() be called at runtime?

A. 0
B. 1
C. 2
D. Not enough information to tell

i>Clicker #3

```
if(foo(count1) < 5 && foo(count2) < 49) {
    cout << "Within range";
}
else {
    cout << "Out of range";
}
```

How many times will foo() be called at runtime?

A. 0
B. 1
C. 2
D. Not enough information to tell

i>Clicker #4

```
if(foo(count1) < 5 || foo(count2) < 49) {
    cout << "Within range";
}
else {
    cout << "Out of range";
}
```

How many times will foo() be called at runtime?

A. 0
B. 1
C. 2
D. Not enough information to tell

i>Clicker #4

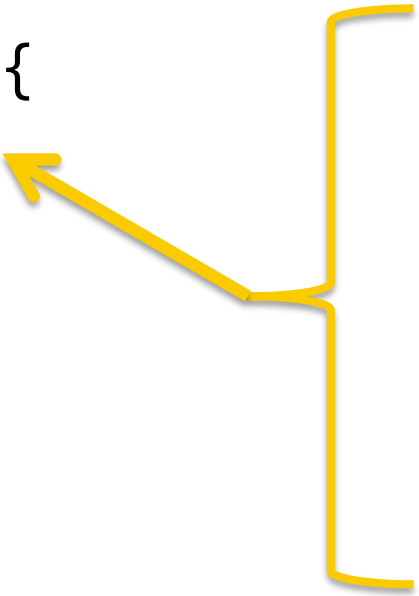
```
if(foo(count1) < 5 || foo(count2) < 49) {
    cout << "Within range";
}
else {
    cout << "Out of range";
}
```

How many times will foo() be called at runtime?

A. 0
B. 1
C. 2
D. Not enough information to tell

if-else is itself a statement

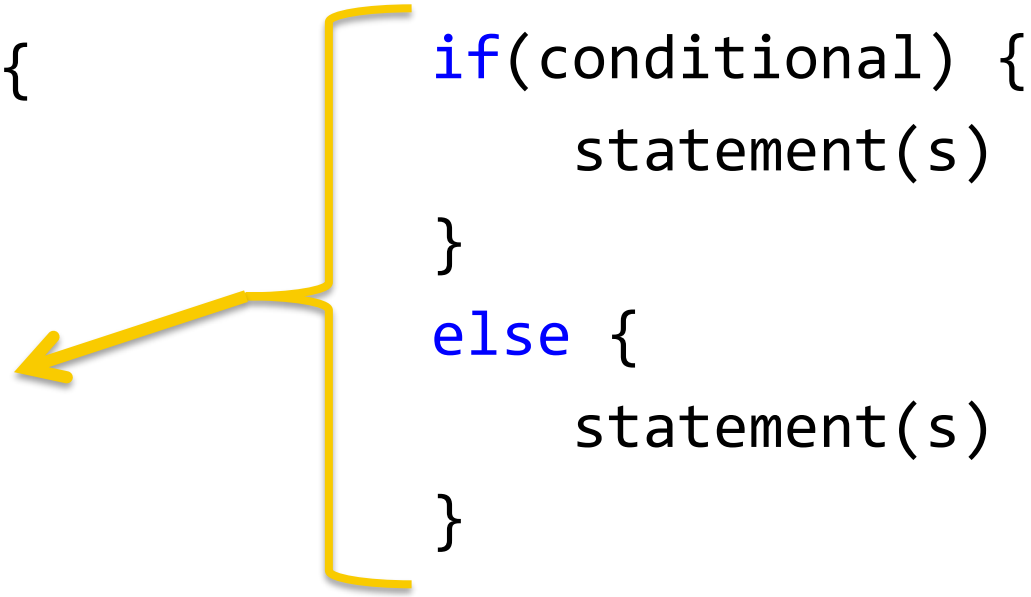
```
if(conditional) {
    statement(s)
}
else {
    statement(s)
}
```



```
if(conditional) {
    statement(s)
}
else {
    statement(s)
}
```

if-else is itself a statement

```
if(conditional) {
     statement(s)
}
else {
     statement(s)
}
```



```
if(conditional) {
     statement(s)
}
else {
     statement(s)
}
```

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
}

if(grade < 80 && grade >= 60) {
    cout << "middle";
}

if(grade < 60) {
    cout << "low";
}
```

Poor style
More typing
More chance of errors
Redundancy

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
} else {
    // Executes when grade < 80
}

if(grade < 80 && grade >= 60) {
    cout << "middle";
}

if(grade < 60) {
    cout << "low";
}
```

else represents the complement of the condition of its associated **if**

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
} else {
    // Executes when grade < 80
}
```

```
if(grade < 80 && grade >= 60) {
    cout << "middle";
}

if(grade < 60) {
    cout << "low";
}
```

else represents the complement of the condition of its associated **if**

These if statements apply when grade < 80

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
} else {
    // Executes when grade < 80
    if(grade < 80 && grade >= 60) {
        cout << "middle";
    }

    if(grade < 60) {
        cout << "low";
    }
}
```

else represents the complement of the condition of its associated **if**

These if statements apply when grade < 80

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
} else {
    if(grade < 80 && grade >= 60) {
        cout << "middle";
    }

    if(grade < 60) {
        cout << "low";
    }
}
```

Remove redundant conditions

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
} else {
    if(grade >= 60) {
        cout << "middle";
    }

    if(grade < 60) {
        cout << "low";
    }
}
```

Apply again to third if statement

if-else is itself a statement

```
if(grade >= 80) {
    cout << "high";
} else {
    if(grade >= 60) {
        cout << "middle";
    } else {
        cout << "low";
    }
}
```

Cascading if-else if-else in C++

```
if(condition1) {
    statement(s)
}
else if(condition2) {
    statement(s)
}
…
…
else {
    statement(s)
}
```

Exactly one statement is executed:
first one for which (condition) is true, or
else if all (condition) are false.

Cascading if-else if-else in C++

Nested if-else statements

```
if(grade >= 80) {
    cout << "high";
} else {
    if(grade >= 60) {
        cout << "middle";
    } else {
        cout << "low";
    }
}
```

Cascading if-else if-else

```
if(grade >= 80) {
    cout << "high";
} else if(grade >= 60) {
    cout << "middle";
} else {
    cout << "low";
}
```

i>Clicker #5

```
if(grade >= 80) {
    cout << "high";
} else {
    if(grade >= 60) {
        cout << "middle";
    } else {
        cout << "low";
    }
}
```

```
if(grade >= 60) {
    cout << "middle";
} else if(grade >= 80) {
    cout << "high";
} else {
    cout << "low";
}
```

Do these blocks of code do the same thing?

A. Yes
B. No

i>Clicker #5

```
if(grade >= 80) {
    cout << "high";
} else {
    if(grade >= 60) {
        cout << "middle";
    } else {
        cout << "low";
    }
}
```

```
if(grade >= 60) {
    cout << "middle";
} else if(grade >= 80) {
    cout << "high";
} else {
    cout << "low";
}
```

Order matters

Do these blocks of code do the same thing?

A. Yes

B. No

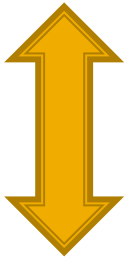
Today

Compound assignment
Increment
Decrement
Loops

Compound Assignment Operators

- += *= -= /= %=

variable += expression



variable = variable + (expression)

Compound Assignment

```
int  x;
x = 2;
x += 5;
x -= 2;
x *= 3;
```


```
int  x;
x = 2;
x = x + 5;
x = x - 2;
x = x * 3;
```

i>Clicker #6

```
int x = 4;
x *= 3 + 1;
cout << x;
```

What does this code print out?

A. 16
B. 13
C. 3
D. 4

i>Clicker #6

```
int x = 4;
x *= 3 + 1;
cout << x;
```

What does this code print out?

A. 16
B. 13
C. 3
D. 4

```
x *= 3 + 1;
```

```
x = x * (3 + 1);
```

NOT `x = x * 3 + 1;`

Increment Decrement Operators

- Increment: `i++` `++i`  `i = i + 1`
- Decrement: `i--` `--i`  `i = i - 1`

i++ vs ++i

Statements

```
i++;
```

```
i = i + 1;
```

```
++i;
```

```
i = i + 1;
```

When used in Expressions

```
cout << i++;
```

```
cout << i;
i = i + 1;
```

```
cout << ++i;
```

```
i = i + 1;
cout << i;
```

i++ vs ++i

No Problem Ever

```
i++;
```

```
i = i + 1;
```

```
++i;
```

```
i = i + 1;
```

Be Extremely Cautious

```
cout << i++;
```

```
cout << i;
i = i + 1;
```

```
cout << ++i;
```

```
i = i + 1;
cout << i;
```

i>Clicker #7

```
int i = 4;
cout << i++;
cout << ++i;
```

What does this code print out?

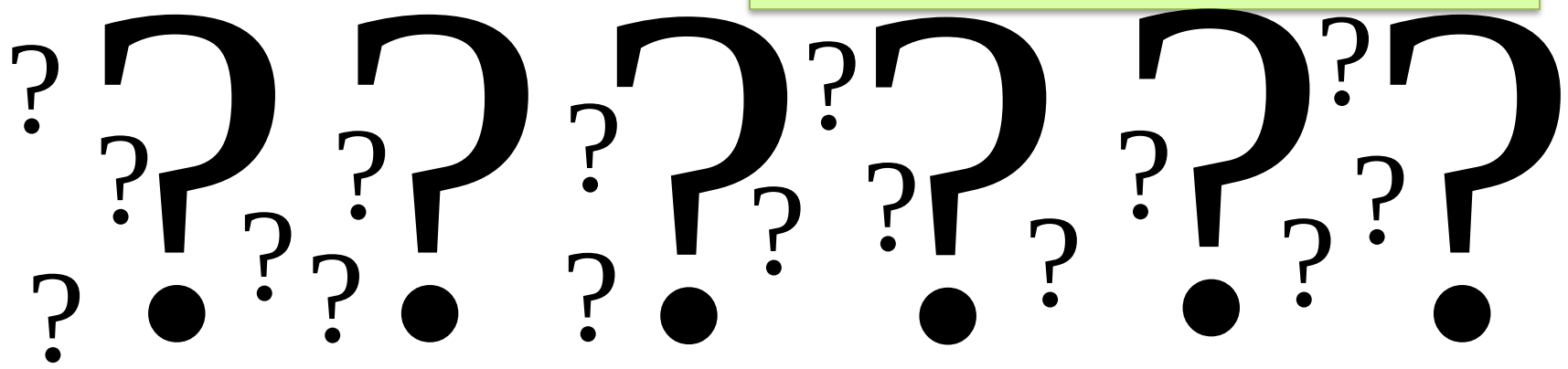
A. 55
B. 56
C. 44
D. 46
E. 45

i>Clicker #7

```
int i = 4;
cout << i++;
cout << ++i;
```

What does this code print out?

A. 55
B. 56
C. 44
D. 46
E. 45



Loops

What loops are and how they are used
How to build various loops

Computers Excel @

- Doing same thing over and over again
- Software never gets tired

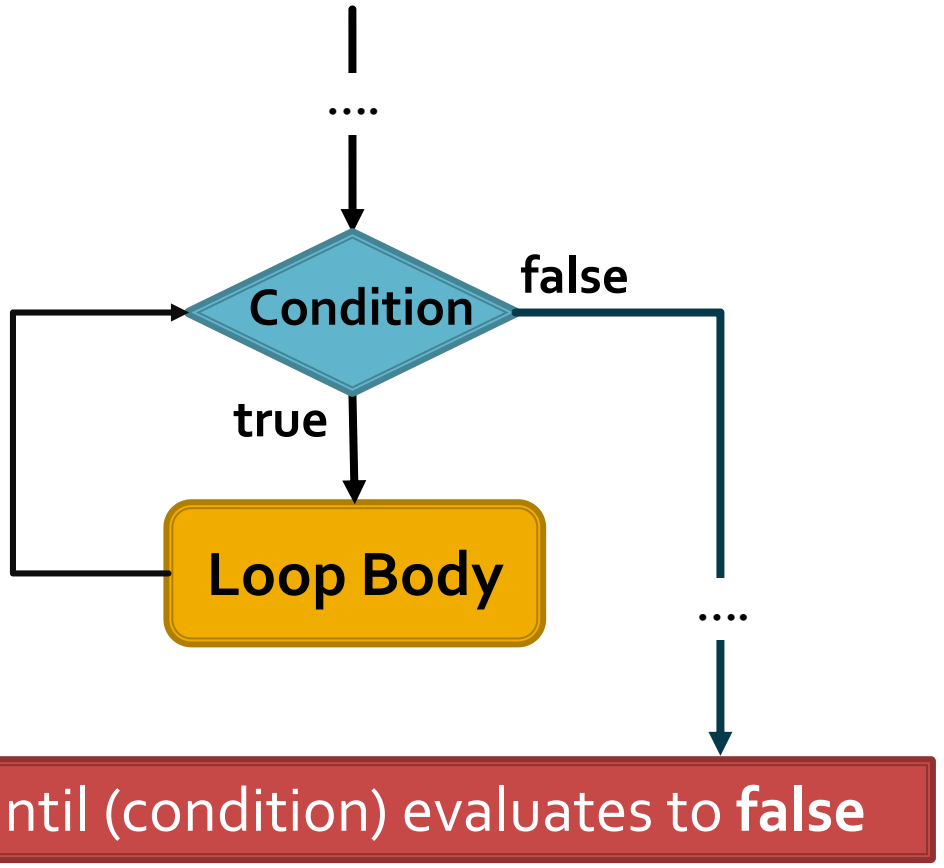
What's a loop?

- a block of code that is executed more than once in a row

What's a loop?

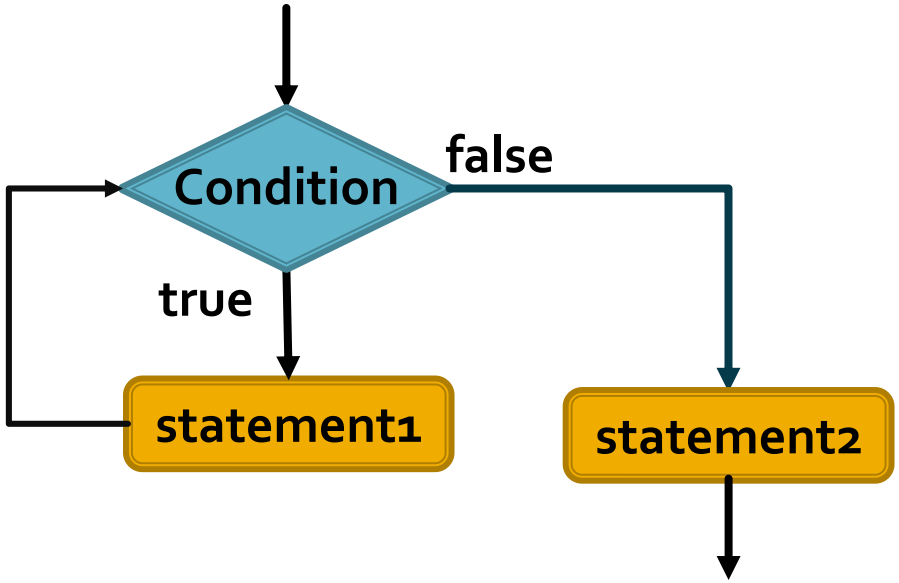
- A block of code that is executed more than once in a row

```
while(condition) {
    statement(s)
}
```



Loop body

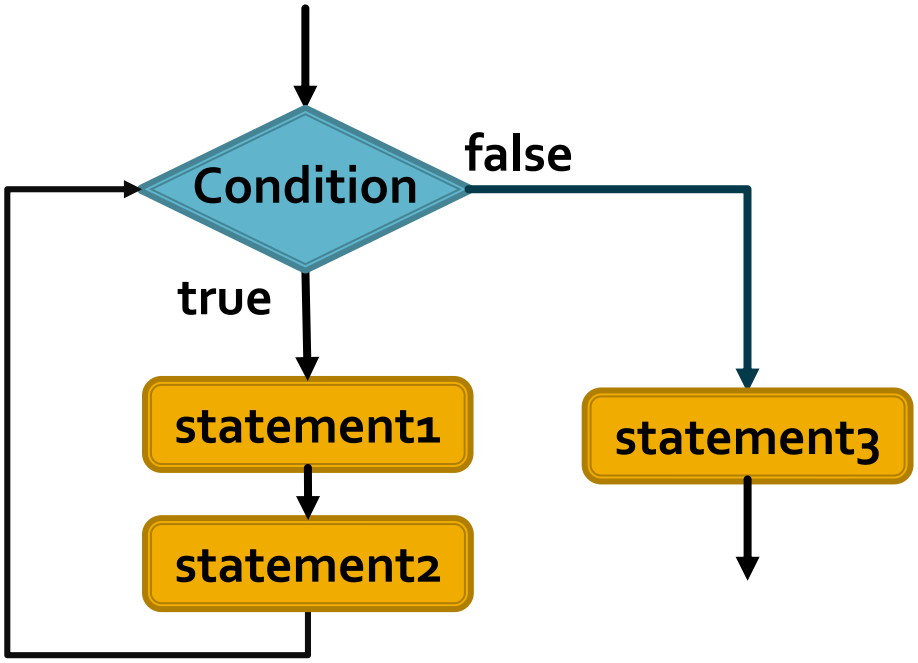
```
while(condition) {
    statement1;
}

statement2;
```



A simple statement loop body

Loop body

```
while(condition) {
    statement1;
    statement2;
}

statement3;
```

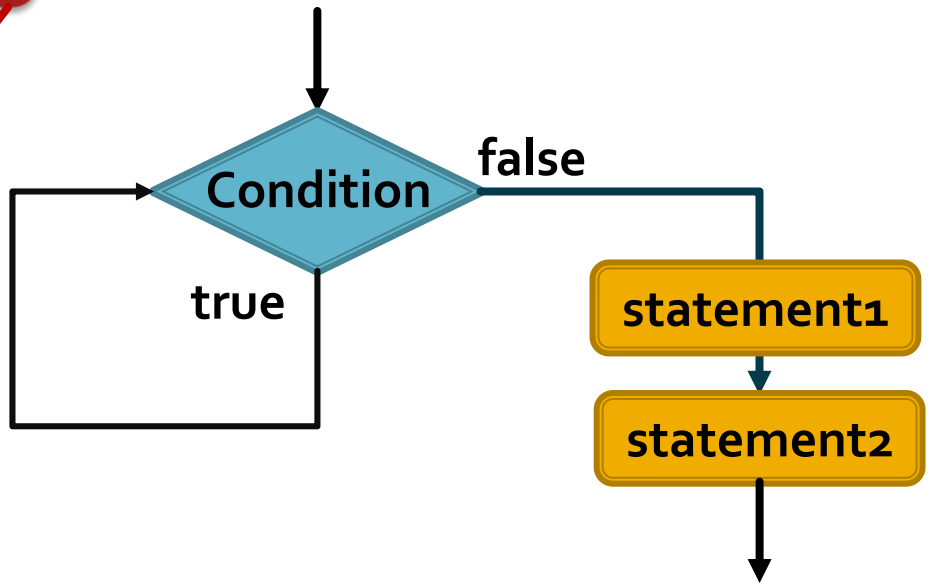


A compound statement loop body

Loop body

notice the ';'

```
while(condition);
    statement1;

statement2;
```



An **empty** statement loop body

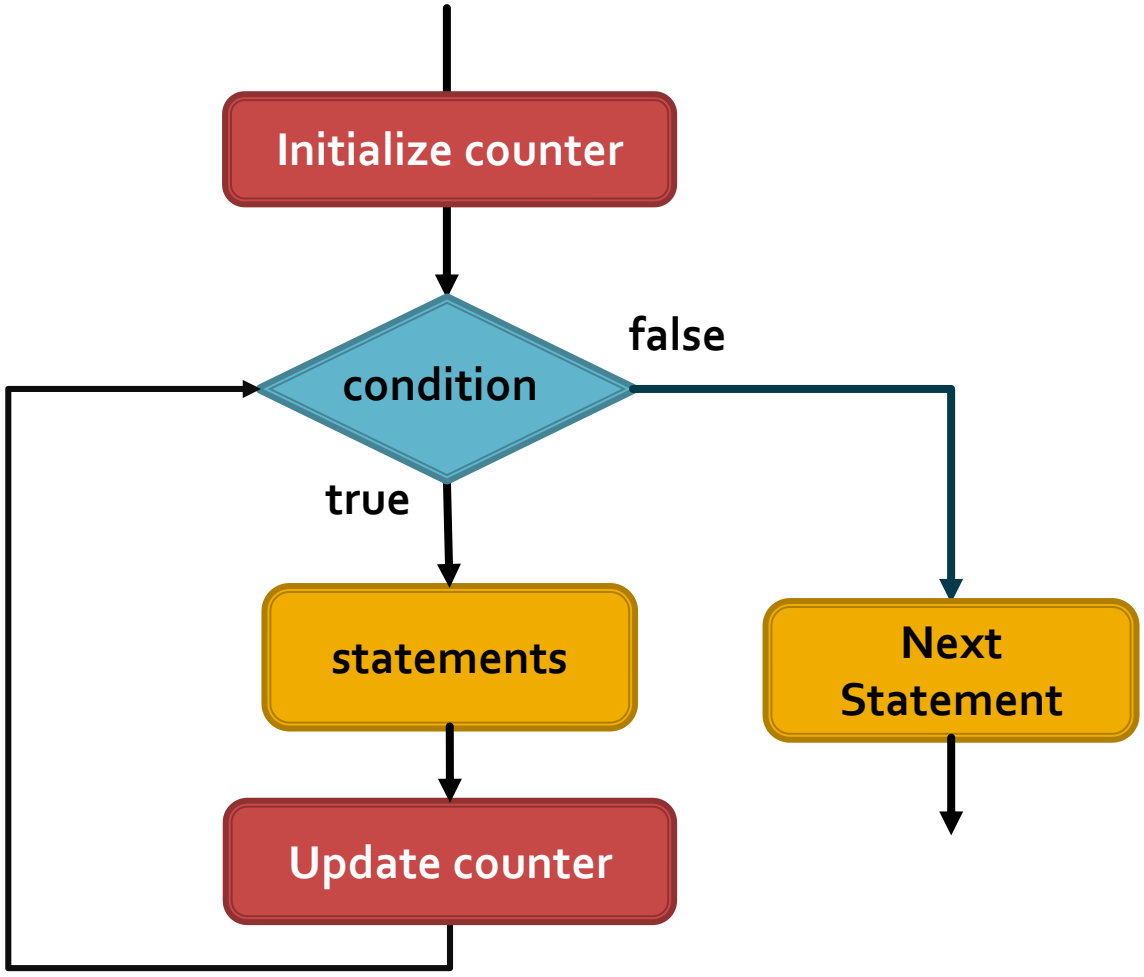
Usually leads to an infinite loop

Uses of while

- Count Controlled Loops
- Event Controlled Loops

Count Controlled Loops

```
// Initialize counter

while(check count) {
     statement1;
     statement2;
     // update counter
}

// Next statement
```



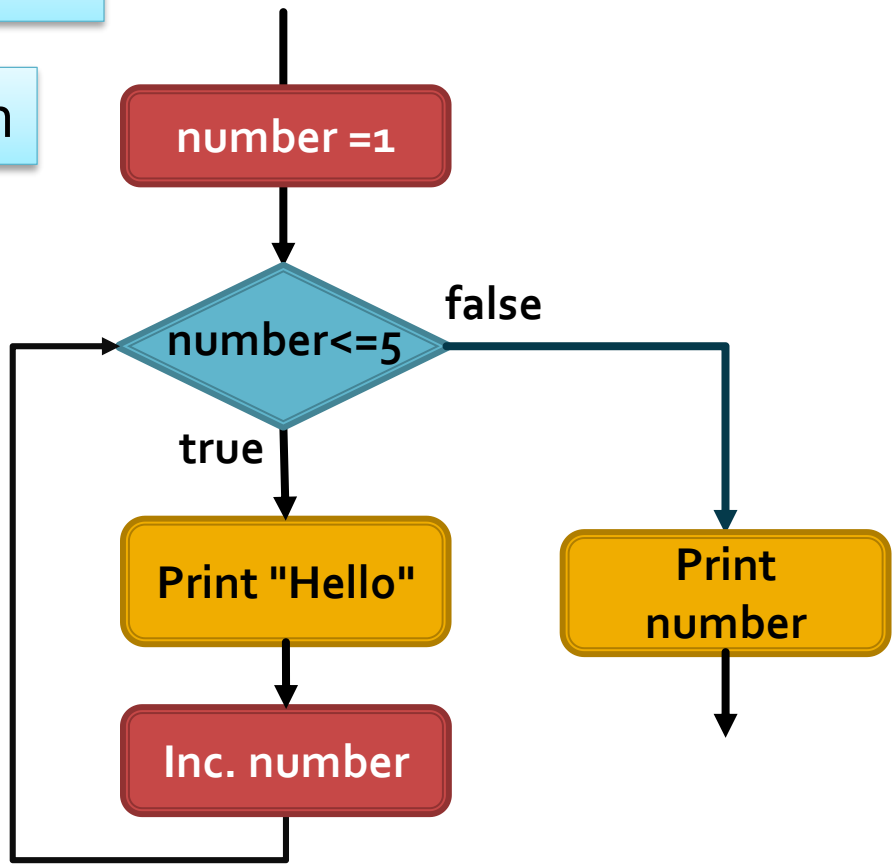
Example

```
int number = 1;

while(number <= 5) {
     cout << "Hello"
          << endl;
     number++;
}

cout << number;
```

Initialization

Condition

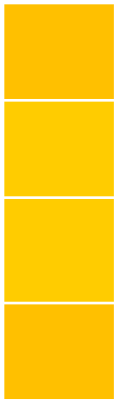
Update



Example

Execution

```
int number = 1;

while(number <= 5) {
     cout << "Hello"
          << endl;
     number++;
}

cout << number;
```

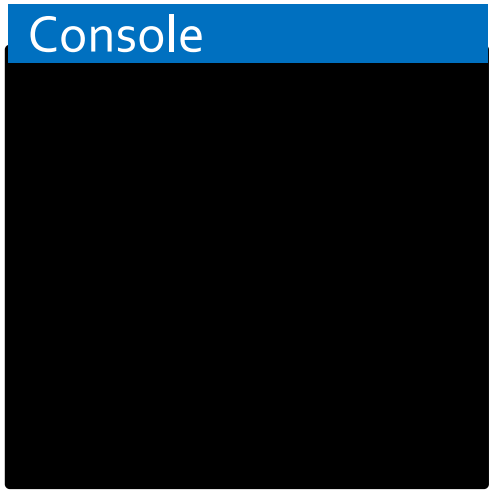


Console

Example

Execution

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```



Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

number

Console

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

Hello

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

Hello

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

Hello

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

number

Console

```
Hello
Hello
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
Hello
Hello
```

Example

```
int number = 1;

while(number <= 5) {
    cout << "Hello"
         << endl;
    number++;
}

cout << number;
```

Execution

Console

```
Hello
Hello
Hello
Hello
Hello
6
```

What is last value to print?

```
int number = 7;

while(number <= 5)
{
    cout << "Hello" << endl;
    number++;
}

cout << number;
```

A) Hello
B) 8
C) 7
D) 6
E) None of the above

What is last value to print?

```
int number = 7;

while(number <= 5)
{
    cout << "Hello" << endl;
    number++;
}

cout << number;
```

A) Hello
B) 8
C) 7
D) 6
E) None of the above

What prints?

```
int number = 1;

while(number <= 5) {
    cout << "Hello" << endl;
}

cout << number;
```

A) 5 "Hello"s
B) 6 "Hello"s
C) 4 "Hello"s
D) 1
E) Infinite # of "Hello"s

What prints?

```
int number = 1;

while(number <= 5) {
    cout << "Hello" << endl;
}

cout << number;
```

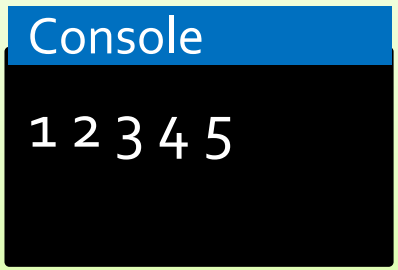
A) 5 "Hello"s
B) 6 "Hello"s
C) 4 "Hello"s
D) 1
E) Infinite # of "Hello"s

No update

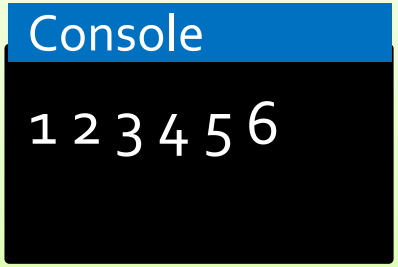
i>Clicker #8

```
int number = 1;

while(number <= 5) {
    cout << number << ' ';
    number++;
}

cout << endl << number;
```

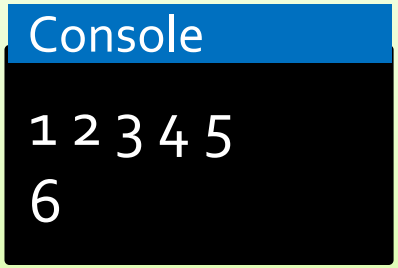
What does this code print out?



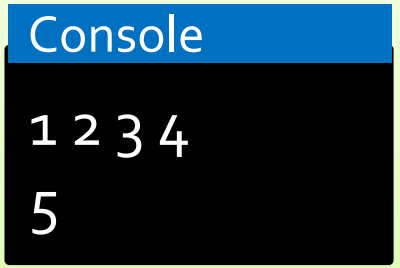
A



B



C

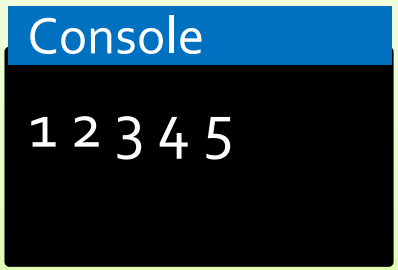


D

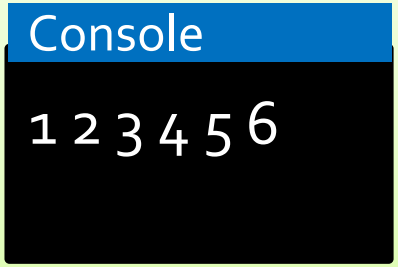
i>Clicker #8

```
int number = 1;

while(number <= 5) {
    cout << number << ' ';
    number++;
}

cout << endl << number;
```

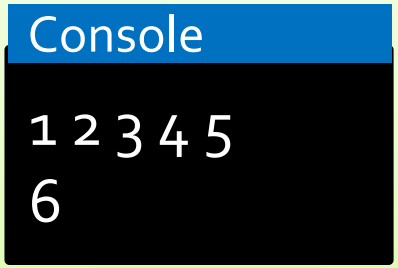
What does this code print out?



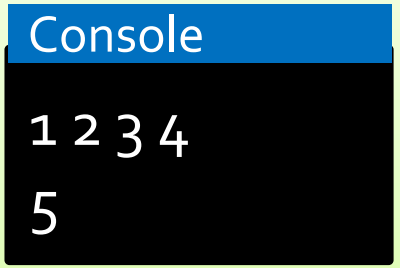
A



B



C



D

i>Clicker #8

```
int number = 1;

while(number < 5) {
    cout << number << ' ';
    number++;
}

cout << endl << number;
```

What does this code print out?

A
```
Console
1 2 3 4 5
```

B
```
Console
1 2 3 4 5 6
```

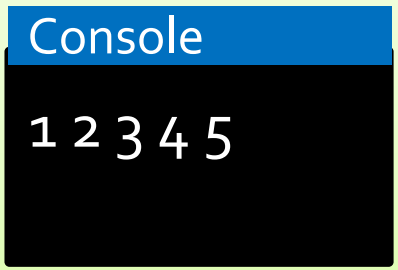
C
```
Console
1 2 3 4 5
6
```

D
```
Console
1 2 3 4
5
```

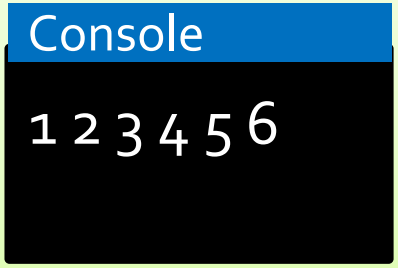
i>Clicker #8

```
int number = 1;

while(number < 5) {
    cout << number << ' ';
    number++;
}

cout << endl << number;
```

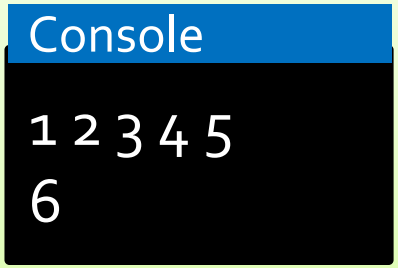
What does this code print out?



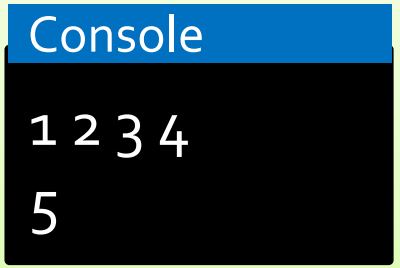
A



B



C



D

What prints?

```
int number = 1;

while(number < 5)
    number++;
    cout << number << ' ';

cout << number;
```

A) 2 3 4 5
B) 1 2 3 4 5
C) 1 2 3
D) 5 5
E) Infinite loop

What prints?

```
int number = 1;

while(number < 5)
    number++;

cout << number << ' ';
cout << number;
```

A) 2 3 4 5
B) 1 2 3 4 5
C) 1 2 3
D) 5 5
E) Infinite loop

What prints?

```
int number = 1;

while(number < 5); {
     number++;
     cout << number << ' ';
}

cout << number;
```

A) 2 3 4 5
B) 1 2 3 4 5
C) 1 2 3
D) 5 5
E) Infinite loop

What prints?

```
int number = 1;

while(number < 5); {
     number++;
     cout << number << ' ';
}

cout << number;
```

notice

A) 2 3 4 5
B) 1 2 3 4 5
C) 1 2 3
D) 5 5
E) Infinite loop

What prints?

```
int x = 1;

while(x <= 100) {
     cout << x << ' ';
     x++;
}
```

A) 1 … 99
B) 1 … 100
C) 1 … 101
D) nothing
E) infinite loop

What prints?

```
int x = 1;

while(x <= 100) {
     cout << x << ' ';
     x++;
}
```

A) 1 … 99
B) 1 … 100
C) 1 … 101
D) nothing
E) infinite loop

What prints?

```
int x = 1;

while(x < 100) {
    cout << x << ' ';
    x++;
}
```

A) 1 … 99
B) 1 … 100
C) 1 … 101
D) nothing
E) infinite loop

What prints?

```
int x = 1;

while(x < 100) {
     cout << x << ' ';
     x++;
}
```

A) 1 … 99
B) 1 … 100
C) 1 … 101
D) nothing
E) infinite loop

What prints?

```
int x = 1;

while(x < 100)
     cout << x << ' ';
     x++;
```

A) 1 … 99
B) 1 … 100
C) 1 … 101
D) nothing
E) infinite loop

What prints?

```
int x = 1;

while(x < 100)
     cout << x << ' ';

x++;
```

A) 1 … 99
B) 1 … 100
C) 1 … 101
D) nothing
E) infinite loop

No update

What prints? Counting down

```
int x = 100;

while(x > 0) {
     cout << x << ' ';
     x--;
}
```

What prints? Counting down

```
int x = 100;

while(x > 0) {
     cout << x << ' ';
     x--;
}
```

100 … 1

What prints? Counting down

```
int x = 100;

while(x >= 0) {
     cout << x << ' ';
     x--;
}
```

What prints?

```
int x = 100;

while(x >= 0) {
     cout << x << ' ';
     x--;
}
```

100 ... 0

What prints?

```
int x = 100;

while(x >= 0) {
    x--;
    cout << x << ' ';
}
```

What prints?

```
int x = 100;

while(x >= 0) {
    x--;
    cout << x << ' ';
}
```

99 … -1

What prints?

```
int x = 1;

while(x <= 100) {
    if(x % 13 == 0) {
        cout << x << endl;
    }

    x++;
}
```

What prints?

```
int x = 1;

while(x <= 100) {
    if(x % 13 == 0) {
        cout << x << endl;
    }

    x++;
}
```

```
13
26
39
52
65
78
91
```

Prints all integers between 1 and 100 (inclusive) that are divisible by 13

What prints?

```
int x = 13;

while(x <= 100) {
    cout << x << endl;
    x += 13;
}
```

```
13
26
39
52
65
78
91
```

Event Controlled Loops

Event Controlled Loops

- Repeat executing code until some event happens.
  - No counting
- Useful for input validation
  - Keep asking the users to re-enter data as long as the their input is invalid.

Event Controlled Loops

```
int year;

// initialize loop variable
cin >> year;

// Loop until invalid year
while ( year <= 0 ) {
     cout << "Year must be greater than 0"
          << endl
          << "Try again.."
          << endl;

     // update loop control variable
     cin >> year;
}

cout << year << endl;
```

Event Controlled Loops

```
int year;

cin >> year;

// Get and check year
while (cin >> year && year <= 0 ) {
    cout << "Year must be greater than 0"
         << endl
         << "Try again.."
         << endl;
    cin >> year;
}

cout << year << endl;
```

Event Controlled Loops

```
int year;

cin >> year;

// Get and check year
while (cin >> year && year <= 0 ) {
    cout << "Year must be greater than 0"
         << endl
         << "Try again.."
         << endl;
    cin >> year;
}

cout << year << endl;
```

true when cin reads valid value

Event Controlled Loops

```
int year = -1;


while (year <= 0 )
{
        cin >> year;
        cout << "Year must be greater than 0"
             << endl
             << "Try again.."
             << endl;
}
```

initialize loop variable

check loop condition

update loop control variable

do NOT write like this

Prints regardless

Example: What prints?
input: 1 2 3 0 4

```
int x = 0;
int sum = 0;
int count = 0;

while(cin >> x && x != 0) {
    sum += x;
    count++;
}

cout << 1.0 * sum / count;
```

Example: What prints?
input: 1 2 3 0 4

```
int x = 0;
int sum = 0;
int count = 0;

while(cin >> x && x != 0) {
    sum += x;
    count++;
}

cout << 1.0 * sum / count;
```

Ans:

2.0

Example: What prints?
input: 1 2 3 0 4

```
int x = 1;
int sum = 0;
int count = 0;

while(x != 0 && cin >> x) {
    sum += x;
    count++;
}

cout << 1.0 * sum / count;
```

Example: What prints?
input: 1 2 3 0 4

```
int x = 1;
int sum = 0;
int count = 0;

while(x != 0 && cin >> x) {
    sum += x;
    count++;
}

cout << 1.0 * sum / count;
```

Ans:

1.5

Example: What prints?
input: 1 2 3 0 4

```
int x = 0;
int sum = 0;
int count = 0;

cin >> x;
while(x != 0) {
    sum += x;
    count++;
    cin >> x;
}

cout << 1.0 * sum / count;
```

Example: What prints?
input: 1 2 3 0 4

```
int x = 0;
int sum = 0;
int count = 0;

cin >> x;
while(x != 0) {
    sum += x;
    count++;
    cin >> x;
}

cout << 1.0 * sum / count;
```

Ans:

2.0

Example: What prints?
input: 1 2 3 0 4

```
int x = 0;
int sum = 0;
int count = 0;

while(x != 0) {
    sum += x;
    count++;
    cin >> x;
}

cout << 1.0 * sum / count;
```

Example: What prints?
input: 1 2 3 0 4

```
int x = 0;
int sum = 0;
int count = 0;

while(x != 0) {
    sum += x;
    count++;
    cin >> x;
}

cout << 1.0 * sum / count;
```

Ans:

divison by 0

Common Loop Errors

- Off by one
  - x < 500
  - need: x <= 500
- Infinite Loop
  - Forgetting to update count
  - Inside the loop

That's all folks

On your own:

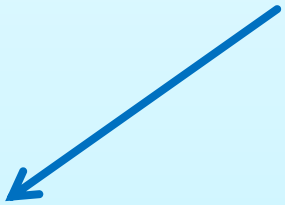
See following slides for examples

Example 1: Average Salary

- don't know how many employees apriori
  - can't be "count" controlled
- decide on "sentinel" value – way to stop the loop
  - positive?
  - 0?
  - negative?

```
int count = 0;
double sum = 0;
double salary;

cin >> salary;
while (salary >= 0) {
   sum = sum + salary;
   count++;
   cin >> salary;
}
if (count > 0) {
   cout << "Average is: " << sum / count;
} else {
   cout << "No data" << endl;
}
```

Not many people will pay to work
But some will work for nothing



```
int count = 0;
double sum = 0;
double salary;

cin >> salary;
while (cin >> salary && salary >= 0) {
   sum = sum + salary;
   count++;
   cin >> salary;
}
if (count > 0) {
   cout << "Average is: " << sum / count;
} else {
   cout << "No data" << endl;
}
```

```
int count = 0;
double sum = 0;
double salary;


while (cin >> salary && salary >= 0) {
   sum = sum + salary;
   count++;

}
if (count > 0) {
   cout << "Average is: " << sum / count;
} else {
   cout << "No data" << endl;
}
```

When a negative is entered

```
int count = 0;
double sum = 0;
double salary;


cin >> salary;
while (cin >> salary && salary >= 0) {
   sum = sum + salary;
   count++;
   cin >> salary;
}
if (count > 0) {
   cout << "Average is: " << sum / count;
} else {
   cout << "No data" << endl;
}
```

Can be written another way

```
int count = 0;
double sum = 0;
double salary;


cin >> salary;
while (cin >> salary && salary >= 0) {
   sum = sum + salary;
   count++;
   cin >> salary;
}
if (count > 0) {
   cout << "Average is: " << sum / count;
} else {
   cout << "No data" << endl;
}
```

Can be written another way

```
sum += salary;
```

Example 2: Average Number

- Don't know how many numbers apriori
- Numbers can be
  - positive
  - negative
  - 0
- Can stop loop by entering a non-number

```
int count = 0;
double total = 0;
double number;

while (cin >> number) {
  total += number;
  count++;
}

if (count > 0) {
   cout << "Average is: " << total / count;
} else {
   cout << "No data" << endl;
}
```

```
int count = 0;
double total = 0;
double number;

while (cin >> number) {
  total += number;
  count++;
}

if (count > 0) {
   cout << "Average is: " << total / count;
} else {
   cout << "No data" << endl;
}
```

alpha will put "cin" into a "fail" state

need to clear before any further input takes place

cin.clear();

Example 3: Sum of first n positive integers

```
// Requires: n >= 0;
// Modifies: nothing
// Effects : returns the sum of the first n
//           positive integers; i.e. 1+2+3+…+n

int sum (int n);
```

Sum of first n positive integers

```
int sum(int n)
{




}
```

Sum of first n positive integers

```
int sum(int n)
{



   while (          )
   {


   }

}
```

Sum of first n positive integers

```
int sum(int n)
{

    int i = 1;

    while ( i <= n )
    {

        i = i + 1;
    }

}
```

Sum of first n positive integers

```
int sum(int n)
{
    int s = 0;
    int i = 1;

    while ( i <= n )
    {
        s = s + i;
        i = i + 1;
    }

}
```

Sum of first n positive integers

```
int sum(int n)
{
    int s = 0;
    int i = 1;

    while ( i <= n )
    {
        s = s + i;
        i = i + 1;
    }
    return s;
}
```

Test Code

```
int main()
{
    cout << sum(3) << "   "
         << sum(1) << "   "
         << sum(10);
…
}
```

Sum of first n positive integers

```
int sum(int n)
{
    int sum = 0;
    int i = 1;

    while ( i <= n )
    {
        sum = sum + i;
        i = i + 1;
    }
    return sum;
}
```

Bad Idea

function name same as variable name

Test Order of 'while' loop

- Test is at the **top**
  - before the "body" of the loop
- Possible for while loop to be done **zero** times!

Example: Finding Primes

- A prime number is evenly divisible only by itself and 1
- How can we use a loop to find a prime number?

Plan of Attack

- Input the potential prime number

Plan of Attack

- Input the potential prime number
- Check all the numbers [2, number) see if they are evenly divisible (%)

Plan of Attack

- Input the potential prime number
- Check all the numbers [2, number) see if they are evenly divisible (%)
- If there is a divisor
  print out number is not prime

Program Development

```
int getValue();
bool isPrime(int number);

int main ( )
{
  int number = getValue();

  if (!isPrime(number)) {
     cout << "Not a prime number "
   } else {
     cout << "Prime number"<< endl;
  }

  return 0;
}
```

Program Development

```
int getValue();
bool isPrime(int number);

int main ( )
{
  int number = getValue();

  if (!isPrime(number)) {
     cout << "Not a prime number "
   } else {
     cout << "Prime number"<< endl;
  }

  return 0;
}
```

Program Development

```
int getValue();
bool isPrime(int number);

int main ( )
{
  int number = getValue();

  if (!isPrime(number)) {
     cout << "Not a prime number "
   } else {
     cout << "Prime number"<< endl;
  }

  return 0;
}
```

Program Development

```
int getValue();
bool isPrime(int number);

int main ( )
{
  int number = getValue();

  if (!isPrime(number)) {
     cout << "Not a prime number "
   } else {
     cout << "Prime number"<< endl;
  }

  return 0;
}
```

Program Development

```
int getValue();
bool isPrime(int number);

int main ( )
{
  int number = getValue();

  if (!isPrime(number)) {
     cout << "Not a prime number "
   } else {
     cout << "Prime number"<< endl;
  }

  return 0;
}
```

```
//Requires: number > 0
//Modifies: nothing
//Effects:  returns true
//          if number is prime
//          false otherwise
bool isPrime(int number)
```

bool isPrime(int number)

```
{
   // start at 2
   // see if it divides number
   // if it does – not a prime
   // if it doesn't increment by 1
   // and repeat
   // stop when divisor >= number




}
```

bool isPrime(int number)

```
{
   // start at 2
   int divisor = 2;

   // see if it divides number
   // if it does – not a prime
   // if it doesn't increment by 1
   // and repeat
   // stop when divisor >= number



}
```

bool isPrime(int number)

```
{
   // start at 2
   int divisor = 2;

   // see if it divides number
   // if it does – not a prime
   if (number % divisor == 0) {
      return false;
   }
   // if it doesn't increment by 1




}
```

bool isPrime(int number)

```
{
   // start at 2
   int divisor = 2;

   // see if it divides number
   // if it does – not a prime
   if (number % divisor == 0) {
      return false;
   }
   // if it doesn't increment by 1
   divisor++;


}
```

bool isPrime(int number)

```
{
   // start at 2
   int divisor = 2;

   // see if it divides number
   // if it does – not a prime
   if (number % divisor == 0) {
      return false;
   }
   // if it doesn't increment by 1
   divisor++;
   // and repeat
   // stop when divisor >= number


}
```

bool isPrime(int number)

```
{
   int divisor = 2;

   // and repeat
   // stop when divisor >= number

   while ( divisor < number) {
      if (number % divisor == 0) {
         return false;
      }
      divisor++;
   }

}
```

bool isPrime(int number)

```
{
   int divisor = 2;

   while ( divisor < number) {
      if (number % divisor == 0) {
         return false;
      }
      divisor++;
   }
   return true;
}
```

bool isPrime(int number)

```
{
   int divisor = 2;

   while ( divisor < number) {
      if (number % divisor == 0) {
         return false;
      }
      divisor++;
   }
   return true;
}
```

bool isPrime(int number)

```
{
   int divisor = 2;

   while ( divisor < number) {
      if (number % divisor == 0) {
         return false;
      }
      divisor++;
   }
   return true;
}
```

bool isPrime(int number)

```
{
   int divisor = 2;

   while ( divisor < number) {
      if (number % divisor == 0) {
         return false;
      }
      divisor++;
   }
   return true;
}
```

testing

```
int getValue();
bool isPrime(int number);

int main ( )
{
  int number = 1;
  while ( number < 100) {
    if (isPrime(number)) {
      cout << number
           << "  is Prime"<< endl;
    }
    number++;
  }
  return 0;
}
```

optimized
bool isPrime(int number)

```
{
    int divisor = 2;
    double sqRoot = sqrt(number);
    while ( divisor <= sqRoot ) {
        if (number % divisor == 0) {
            return false;
        }
        divisor++;
    }
    return true;
}
```