We are 183

L08: Week 5 - Wednesday

Due Soon

- Project 2: due Friday
- Assignment 3: due next Friday
- Exam 1: Less than 2 weeks

Last Time… on EECS 183

Compound Assignment

Increment/decrement

while loops

Compound Assignment

```
int  x;

x = 2;

x += 5;

x -= 2;

x *= 3;
```

Equivalent

```
int  x;

x = 2;

x = x + 5;

x = x - 2;

x = x * 3;
```

i>Clicker #1

```
int x = 4;
x *= x - 1;
cout << x;
```

What does this code print out?

A. 15
B. 12
C. 3
D. Code won't compile

i>Clicker #1

```
int x = 4;
x *= x - 1;
cout << x;
```

What does this code print out?

A. 15
[✓ B. 12]
C. 3
D. Code won't compile

```
x *= x - 1;    x = x * (x - 1);
```

NOT `x = x * x - 1;`

Increment Decrement Operators

- Increment: `i++` `++i` $\Longleftrightarrow$ `i = i + 1`

- Decrement: `i--` `--i` $\Longleftrightarrow$ `i = i - 1`

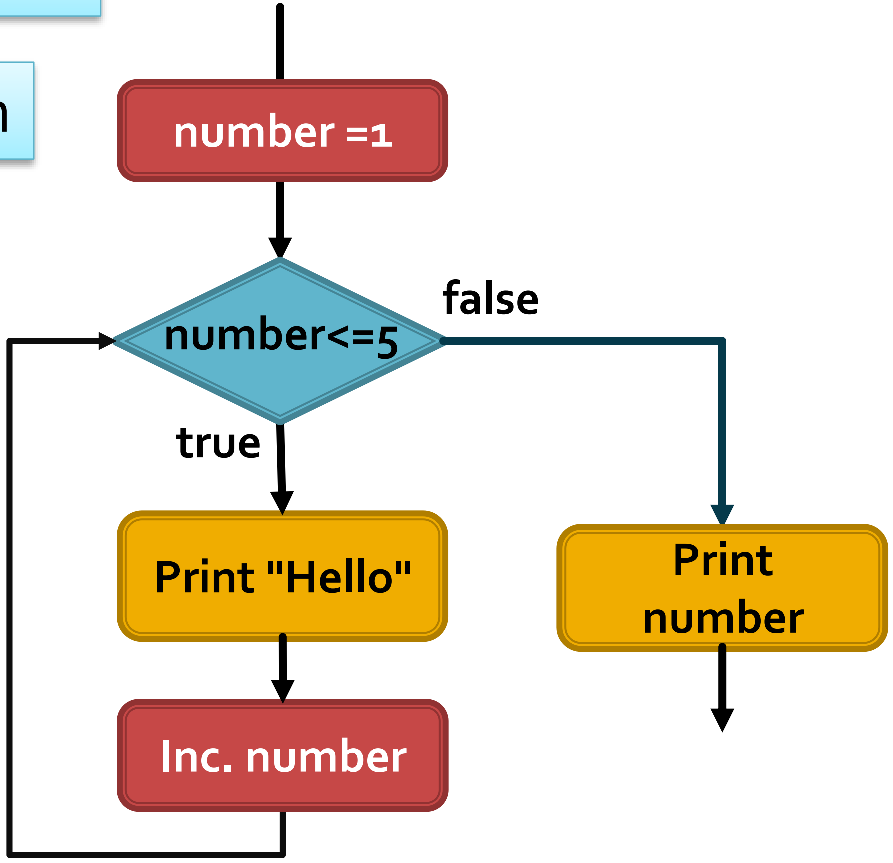
Count Controlled Loops

Initialization

Condition

Update

```
int number = 1;

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

cout << number;
```



i>Clicker #2

```
int number = 1;

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

cout << number;
```

What is the last value printed?

A. 1
B. 5
C. 6
D. None of the above

i>Clicker #2

```
int number = 1;

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

cout << number;
```

What is the last value printed?

A. 1
B. 5
C. 6
[✓ D. None of the above]

i>Clicker #3

```
int number = 10;

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

cout << number;
```

What is the last value printed?

A. 5
B. 4
C. 3
D. None of the above

i>Clicker #3

```
int number = 10;

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

cout << number;
```

What is the last value printed?

A. 5
[✓ B. 4]
C. 3
D. None of the above

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

true when cin reads valid value

```
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;
```

i>Clicker #4

Input: **`3 4 5 0 a 3`**
What does this code print?

A. 3
B. 4
C. Code will result in a runtime error
D. Code will go into an infinite loop

```
int sum = 0;
int count = 0;
int number;

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

cout << (double)sum / count;
```

i>Clicker #4

Input: **`3 4 5 0 a 3`**

What does this code print?

A. 3

[✓ B. 4]

C. Code will result in a runtime error

D. Code will go into an infinite loop

```
int sum = 0;
int count = 0;
int number;

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

cout << (double)sum / count;
```

Clearing a Fail State

Input: `3 4 5 a 3`

A non-number will put cin into fail state

```
int sum = 0;
int count = 0;
int number;

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

cout << (double)sum / count;
```

Clearing a Fail State

Input: `3 4 5 a 3`

```
int sum = 0;
int count = 0;
int number;

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

cout << (double)sum / count;
```

A non-number will put `cin` into fail state

Fail state must be checked for by calling `cin.fail()`, cleared by calling `cin.clear()`

Clearing a Fail State

Input: 3 4 5 a 3

```
int sum = 0;
int count = 0;
int number;

while (cin >> number && number != 0) {
    sum += number;
    count++;
}
cout << (double)sum / count;
if (cin.fail()) {
    cin.clear();
    string str;
    getline(cin, str);
}
```

A non-number will put `cin` into fail state

Fail state must be checked for by calling `cin.fail()`, cleared by calling `cin.clear()`

Clearing a Fail State

Input: 3 4 5 a 3

```
int sum = 0;
int count = 0;
int number;

while (cin >> number && number != 0) {
    sum += number;
    count++;
}
cout << (double)sum / count;
if (cin.fail()) {
    cin.clear();
    string str;
    getline(cin, str);
}
```

A non-number will put cin into fail state

getline() used to get rid of offending input

What Happens?

Input: `0 4 5 a 3`

```
int sum = 0;
int count = 0;
int number;

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

cout << (double)sum / count;
```

What Happens?

Input: 0 4 5 a 3

```
int sum = 0;
int count = 0;
int number;

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

cout << (double)sum / count;
```

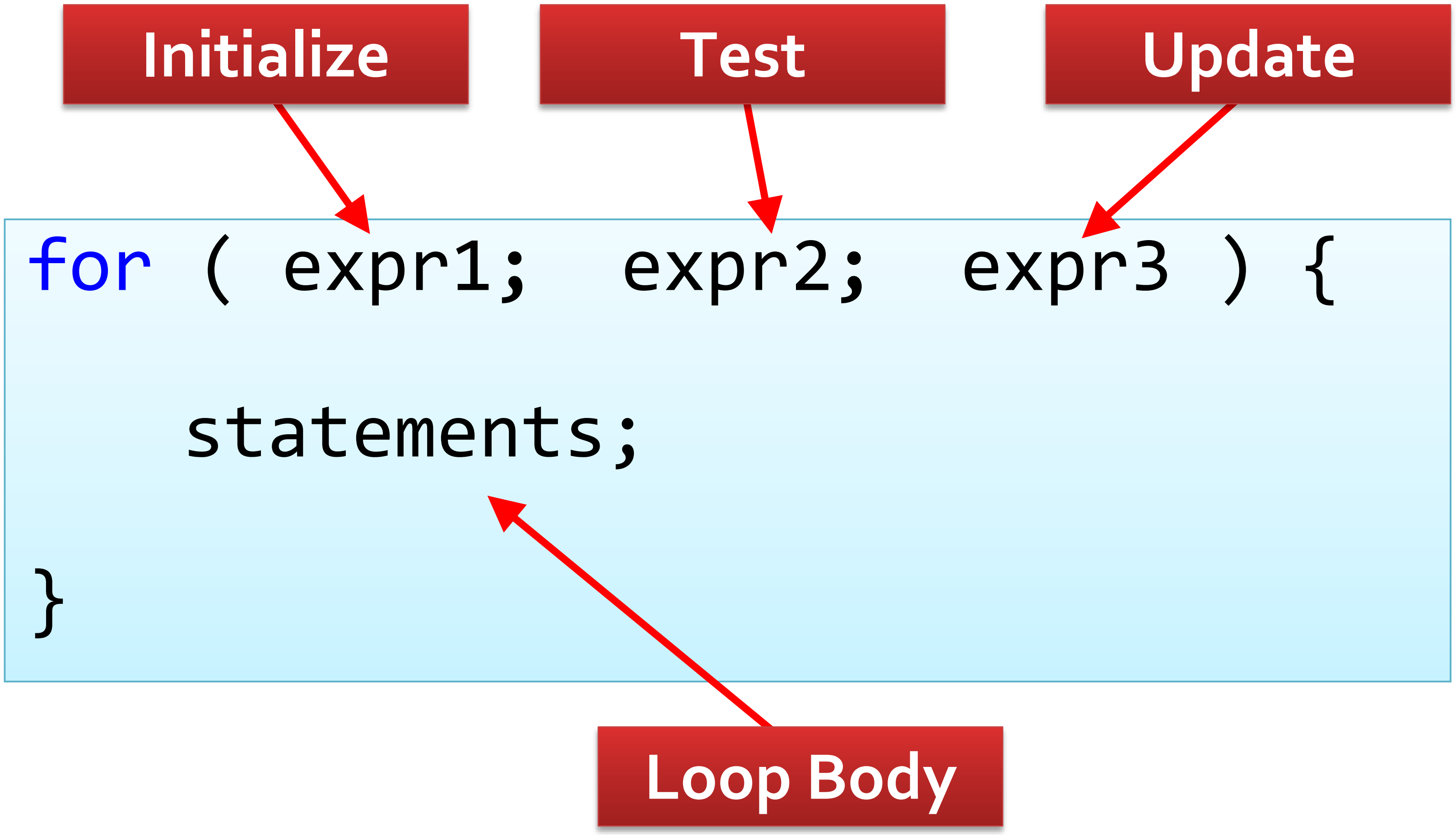
Today

for loops
Nested loops
Strings
Header files

Syntax of *for* loop

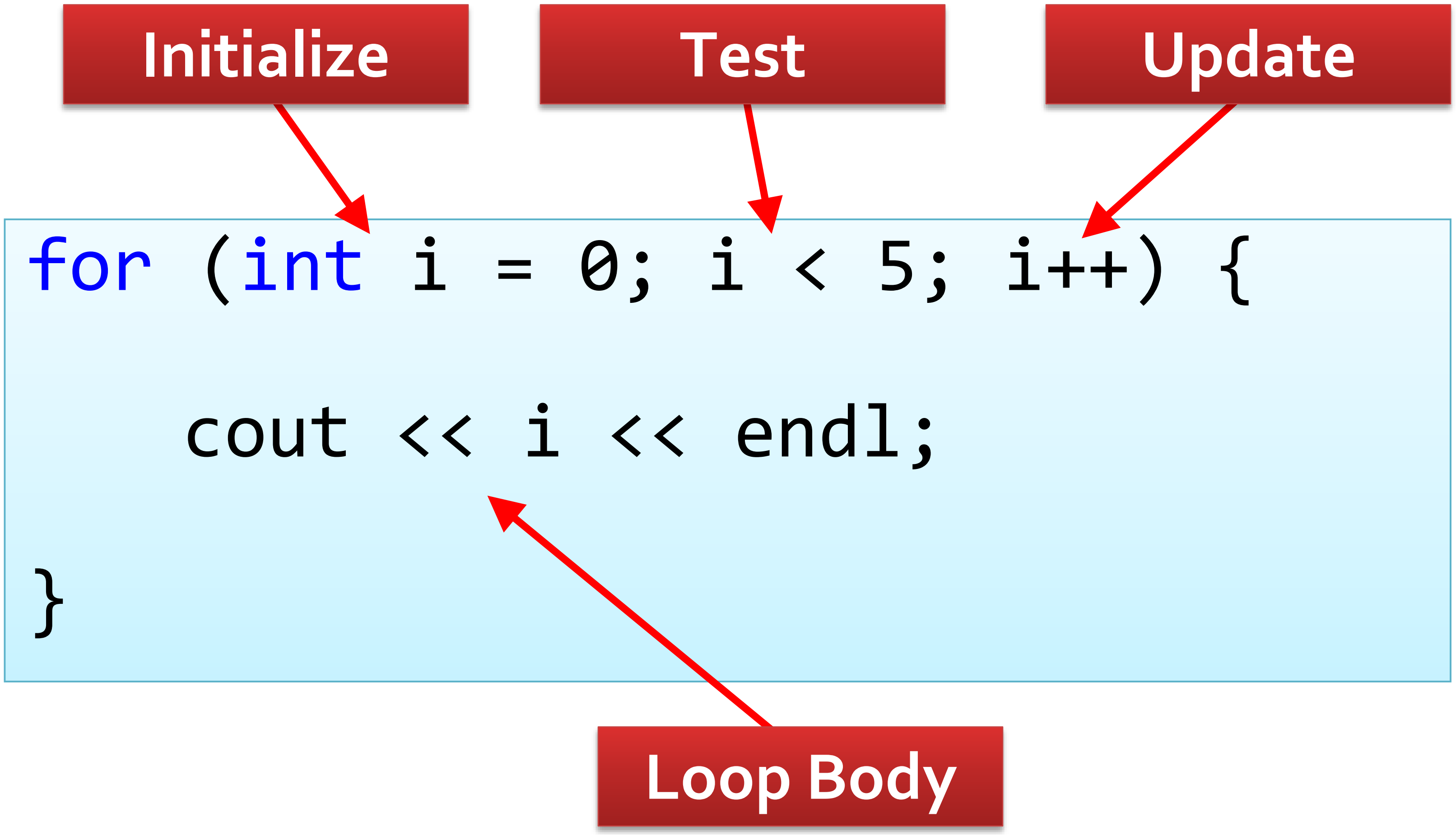
```
for ( expr1;  expr2;  expr3 ) {

    statements;

}
```

Syntax of *for* loop



```
for ( expr1;  expr2;  expr3 ) {

     statements;

}
```

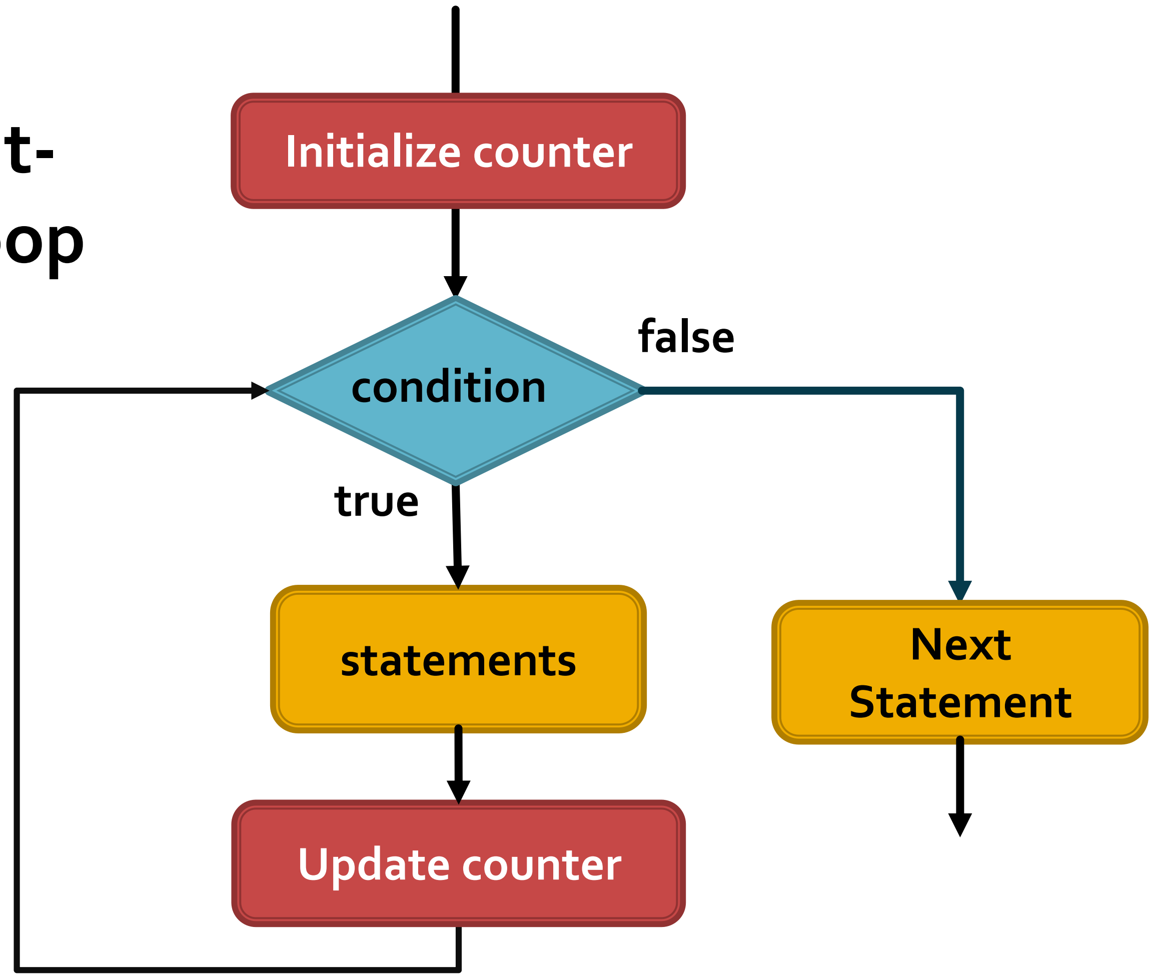
Syntax of *for* loop



```
for (int i = 0; i < 5; i++) {

    cout << i << endl;

}
```

Count Controlled Loops

Basic for loop is equivalent to count-controlled while loop



while loop vs. *for* loop

```
count = 1;
while (count <= 5) {
    square = count * count;
    cout << count << "  " << square << endl;
    count++;
}
```

```
for (count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  " << square << endl;
}
```

while loop vs. *for* loop

```
count = 1;
while (count <= 5) {
    square = count * count;
    cout << count << "  " << square << endl;
    count++;
}
```

```
for (count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  " << square << endl;
}
```

while loop vs. *for* loop

```
count = 1;
while (count <= 5) {
    square = count * count;
    cout << count << "  " << square << endl;
    count++;
}
```

```
for (count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  " << square << endl;
}
```

while loop vs. *for* loop

```
count = 1;
while (count <= 5) {
    square = count * count;
    cout << count << "  " << square << endl;
    count++;
}
```

```
for (count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  " << square << endl;
}
```

Declaring Variable in *for* Initialization

```
int count = 1;
while (count <= 5) {
    square = count * count;
    cout << count << "  " << square << endl;
    count++;
}
```

Counter must be declared outside loop

```
for (int count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  " << square << endl;
}
```

Counter can be declared in for initialization

Variable Scope

Scope of count

```
...
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << " " << square << endl;
}
...
```

Variable Scope

Scope of count

```
...
int count = 1;

for (count = 1; count <= 5; count++) {
   int square = count * count;
   cout << count << " " << square << endl;
}
...
```

If **count** is declared outside the for loop, its scope changes.

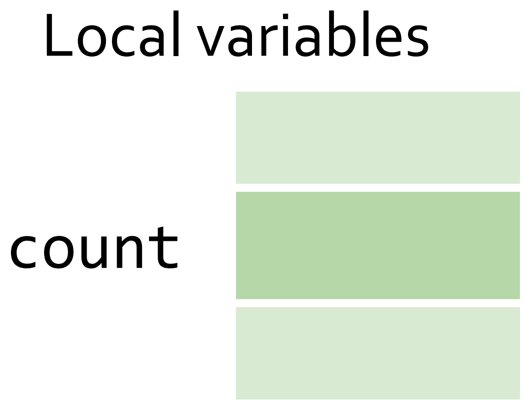
Example: *for* loop

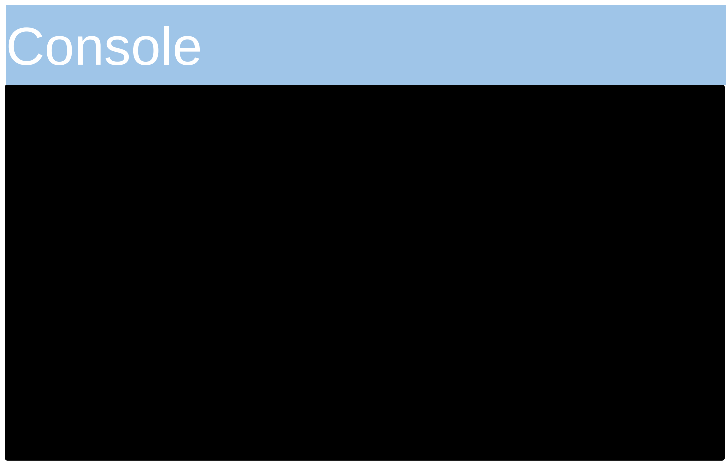
```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Example: *for* loop

initialization

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

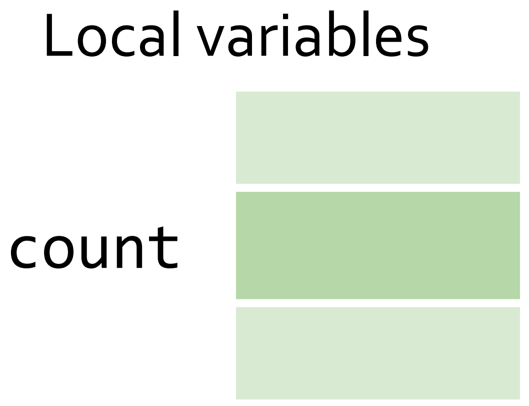


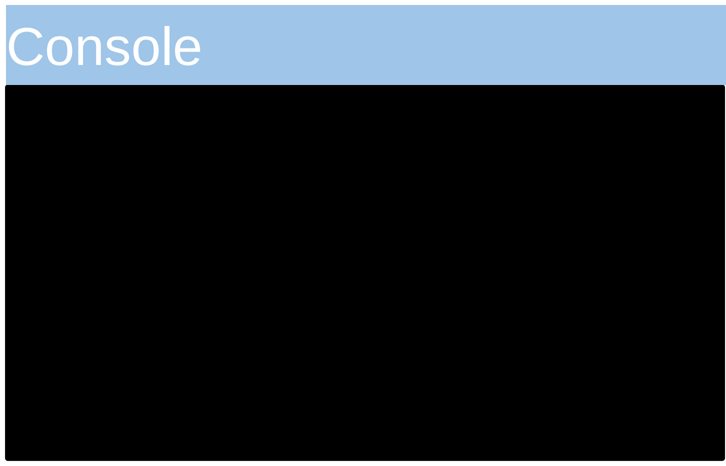


Example: *for* loop

initialization **condition**

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```



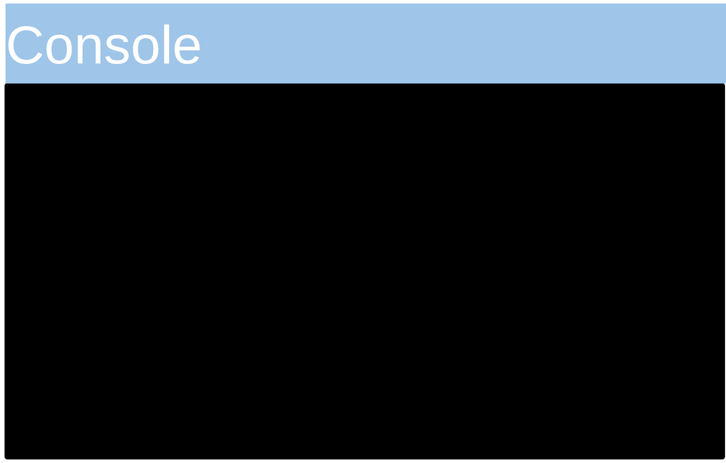


Example: *for* loop

initialization **condition** **update**

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables



Example: *for* loop

initialization   condition   update

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

body

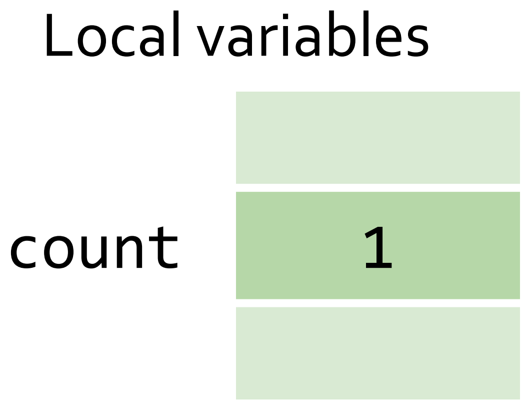
Local variables

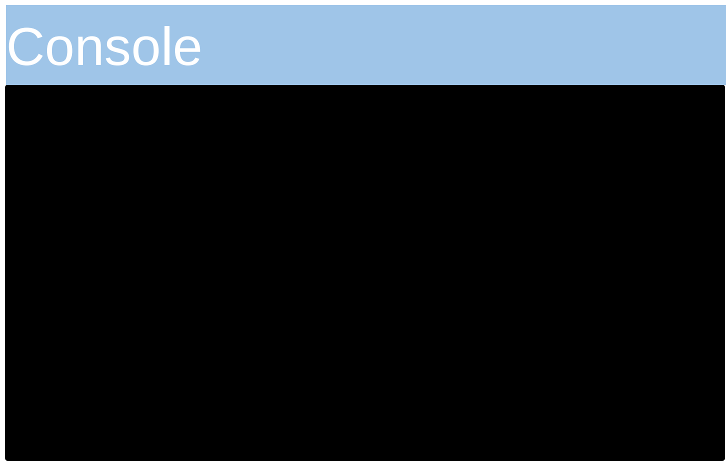
Console

Example: *for* loop

initialize count to 1

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```





Example: *for* loop

check condition for true

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	1

Console

Example: *for* loop

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

execute the body of the loop

Local variables

count	1
square	1

Console

```
1 1
```

Example: *for* loop

update loop control variable

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	2
square	1

Console

```
1 1
```

Example: *for* loop

check condition for true

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	2
square	1

Console

```
1 1
```

Example: *for* loop

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

execute the body of the loop

Local variables

count	2
square	4

Console

```
1 1
2 4
```

Example: *for* loop

update loop control variable

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	3
square	4

Console

```
1 1
2 4
```

Example: *for* loop

check condition for true

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	3
square	4

Console

```
1 1
2 4
```

Example: *for* loop

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

execute the body of the loop

Local variables

count	3
square	9

Console

```
1 1
2 4
3 9
```

Example: *for* loop

update loop control variable

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	4
square	9

Console

```
1 1
2 4
3 9
```

Example: *for* loop

check condition for true

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	4
square	9

Console

```
1 1
2 4
3 9
```

Example: *for* loop

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

execute the body of the loop

Local variables

count	4
square	16

Console

```
1 1
2 4
3 9
4 16
```

Example: *for* loop

update loop control variable

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	5
square	16

Console

```
1 1
2 4
3 9
4 16
```

Example: *for* loop

check condition for true

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	5
square	16

Console

```
1 1
2 4
3 9
4 16
```

Example: *for* loop

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

execute the body of the loop

Local variables

count	5
square	25

Console

```
1 1
2 4
3 9
4 16
5 25
```

Example: *for* loop

update loop control variable

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	6
square	25

Console

```
1 1
2 4
3 9
4 16
5 25
```

Example: *for* loop

check condition for true

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

count	6
square	25

Console

```
1 1
2 4
3 9
4 16
5 25
```

Example: *for* loop

variables declared inside of the loop are deleted

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

~~count~~	~~6~~
~~square~~	~~25~~

Console

```
1 1
2 4
3 9
4 16
5 25
```

Example: *for* loop

continue execution

```
for (int count = 1; count <= 5; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```

Local variables

Console

```
1 1
2 4
3 9
4 16
5 25
```

i>Clicker #5

```
int i = 3;

for (i = 0; i < 3; i++) {
    cout << i;
}
```

What prints?

A. 0123
B. 012
C. 3
D. 0
E. Nothing

i>Clicker #5

```
int i = 3;

for (i = 0; i < 3; i++) {
    cout << i;
}
```

What prints?

A. 0123
[✓ B. 012]
C. 3
D. 0
E. Nothing

What if we … add a counter *within* the for loop body?

```
for (int count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "   "<< square << endl;
    count++;
}
```

What if we … add a counter *within* the for loop body?

```
for (int count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  "<< square << endl;
    count++;
}
```

Equivalent while loop

```
int count = 1;
while (count <= 5) {
    square = count * count;
    cout << count << "  " << square << endl;
    count++;
    count++;
}
```

What if we … add a counter *within* the for loop body?

```
for (int count = 1; count <= 5; count++) {
    square = count * count;
    cout << count << "  "<< square << endl;
    count++;
}
```

NOT a good idea!!!

Console

```
1 1
3 9
5 25
```

What if we … add a counter *within* the for loop body?

```
for (int count = 1; count <= 5; count += 2) {
    square = count * count;
    cout << count << "  "<< square << endl;
}
```

Better to just add 2 each iteration

Console

```
1 1
3 9
5 25
```

i>Clicker #6

```
for (int i = 7; i != 0; i -= 2) {
    cout << "Hello" << endl;
}
```

How many times does this print "Hello"?

A. 7
B. 4
C. 3
D. None of the above

i>Clicker #6

```
for (int i = 7; i != 0; i -= 2) {
    cout << "Hello" << endl;
}
```

How many times does this print "Hello"?

A. 7
B. 4
C. 3
D. None of the above

i>Clicker #7

```
int total = 0;
int num = 3;

for (int i = 0; i < num; i++) {
    total += i;
}

cout << i << endl;
```

What prints?

A. 0
B. 2
C. 3
D. None of the above

i>Clicker #7

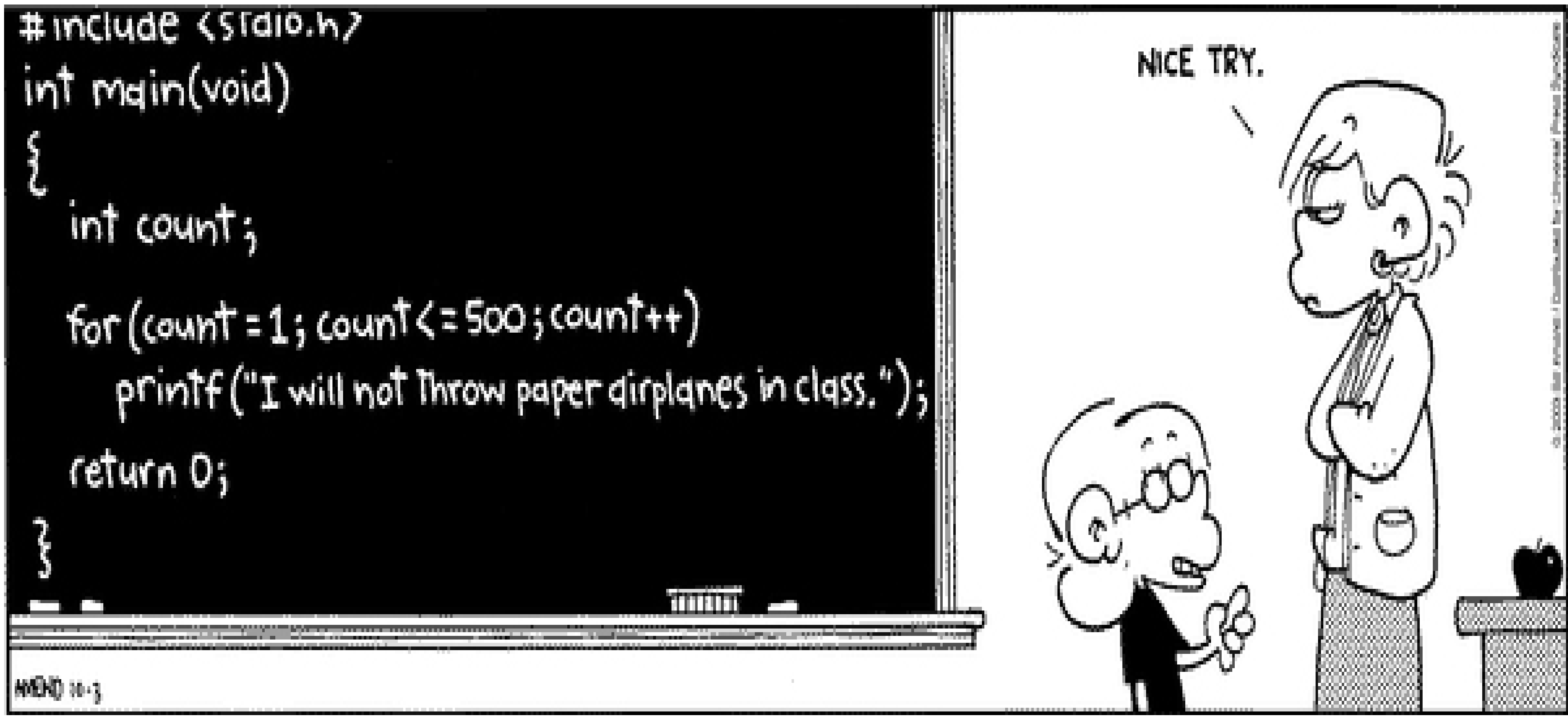
```
int total = 0;
int num = 3;

for (int i = 0; i < num; i++) {
    total += i;
}

cout << i << endl;
```

Scope of `i`

What prints?

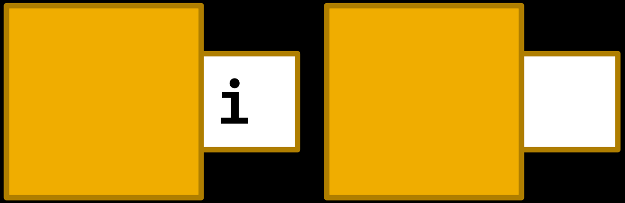
A. 0
B. 2
C. 3
D. None of the above

Count Controlled



```
#include <stdio.h>
int main(void)
{
   int count;

   for(count=1;count<=500;count++)
      printf("I will not Throw paper airplanes in class.");
   return 0;
}
```

Nested *for* loops

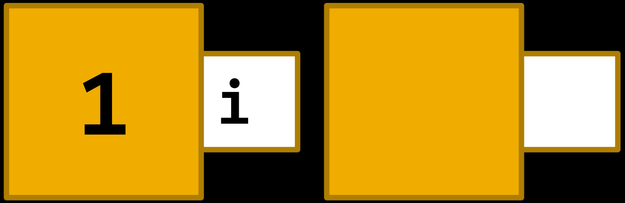


Execution

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

Output

Nested *for* loops

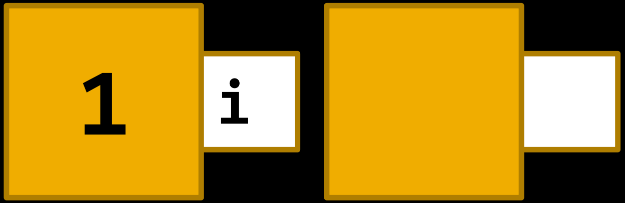


Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

Nested *for* loops



Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

✓

Output

Nested *for* loops

1 i 0 j

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

Nested *for* loops

1	i	0	j

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < ✓i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

Nested *for* loops

1	i	0	j

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

```
*
```

Nested *for* loops

1 i 1 j

Execution

```
for (int i = 1; i < 3; i++) {
  for (int j = 0; j < i; j++) {
     cout << '*';
  }

  cout << endl;
}
```

Output

```
*
```

Nested *for* loops

1 i 1 j

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

```
*
```

Nested *for* loops

1 i

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Execution

Output

```
*
```

Nested *for* loops

2 i

Execution

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

Output

```
*
```

Nested *for* loops

2 i

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

✓

Output

```
*
```

Nested *for* loops

2	i	0	j

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

```
*
```

Nested *for* loops

2 i 0 j

Execution

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

✓

Output

```
*
```

Nested *for* loops

2 i 0 j

Execution

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

Output

```
*
*
```

Nested *for* loops

2 i 1 j

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

Execution

Output

```
*
*
```

Nested *for* loops

2 i

1 j

Execution

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

✓

Output

```
*
*
```

Nested *for* loops

2 i 1 j

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Execution

Output

```
*
**
```

Nested *for* loops

2 i

2 j

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Execution

Output

```
*
**
```

Nested *for* loops

2 i 2 j

Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

```
*
**
```

Nested *for* loops

2 i

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Execution

Output

```
*

**
```

Nested *for* loops

```
3 i
```

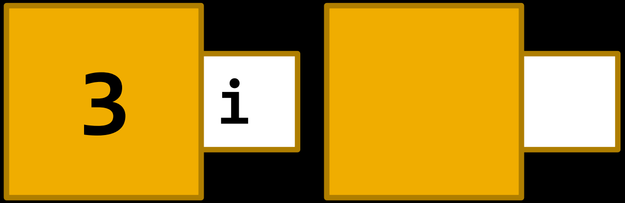
Execution

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
      cout << '*';
   }

   cout << endl;
}
```

Output

```
*
**
```

Nested *for* loops



Execution

```
for (int i = 1; i < 3; i++) {
    for (int j = 0; j < i; j++) {
        cout << '*';
    }

    cout << endl;
}
```

Output

```
*
**
```

Nested *for* loops

```
for (int i = 1; i < 3; i++) {
   for (int j = 0; j < i; j++) {
       cout << '*';
   }

   cout << endl;
}
```

Execution

Output

```
*

**
```

i>Clicker #8

```
for (int i = 0; i < 4; i++) {
    for (int k = i; k > 0; k--) {
        cout << "Hello" << endl;
    }
}
```

How many times does this print "Hello"?

A. 10
B. 6
C. 3
D. None of the above

i>Clicker #8

```
for (int i = 0; i < 4; i++) {
    for (int k = i; k > 0; k--) {
        cout << "Hello" << endl;
    }
}
```

How many times does this print "Hello"?

A. 10
[✓ B. 6]
C. 3
D. None of the above

i>Clicker #9

```
for (char c = 'a'; c < 'e'; c++) {
    cout << c;
}
```

What does this print?

A. abcde
B. abcd
C. Nothing
D. Code won't compile

i>Clicker #9

```
for (char c = 'a'; c < 'e'; c++) {
    cout << c;
}
```

What does this print?

A. abcde
[✓ B. abcd]
C. Nothing
D. Code won't compile

What kind of loop is best?

Is the loop count-controlled?

`for` is usually best

What kind of loop is best?

Is the loop count-controlled?

`for` is usually best

Is the loop event-controlled?

`while` should be used

More on Strings

String = Sequence of chars

- A *sequence* is an ordered grouping of data
- Can all be referenced using the same name
- Elements of the sequence can also be accessed individually
- String literals are defined by double-quotes

```
"String literal"
```

strings in C++

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

int main() {
   string str1 = "Hello";
   string str2 = "World";

   string str3 = str1 + " " + str2;
   cout << str3 << '!' << endl;

   return 0;
}
```

Console Output

```
Hello World!
```

strings in C++

```
#include <string>
```

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

int main() {
   string str1 = "Hello";
   string str2 = "World";

   string str3 = str1 + " " + str2;
   cout << str3 << '!' << endl;

   return 0;
}
```

Console Output

```
Hello World!
```

strings in C++

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

int main() {
    string str1 = "Hello";
    string str2 = "World";

    string str3 = str1 + " " + str2;
    cout << str3 << '!' << endl;

    return 0;
}
```

#include <string>

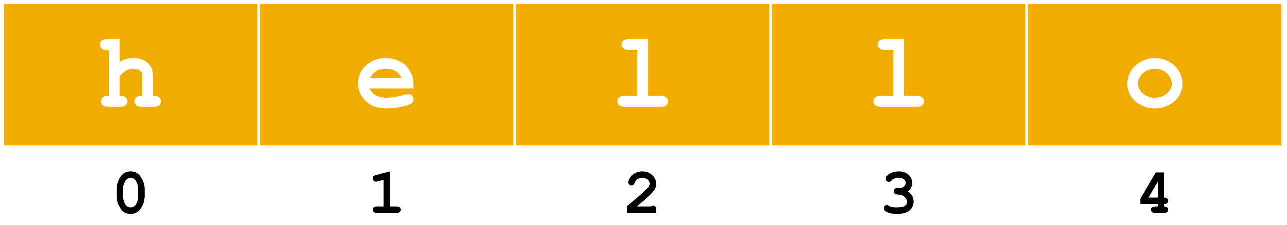
+ operator concatenates strings

Console Output

```
Hello World!
```

chars can be accessed individually

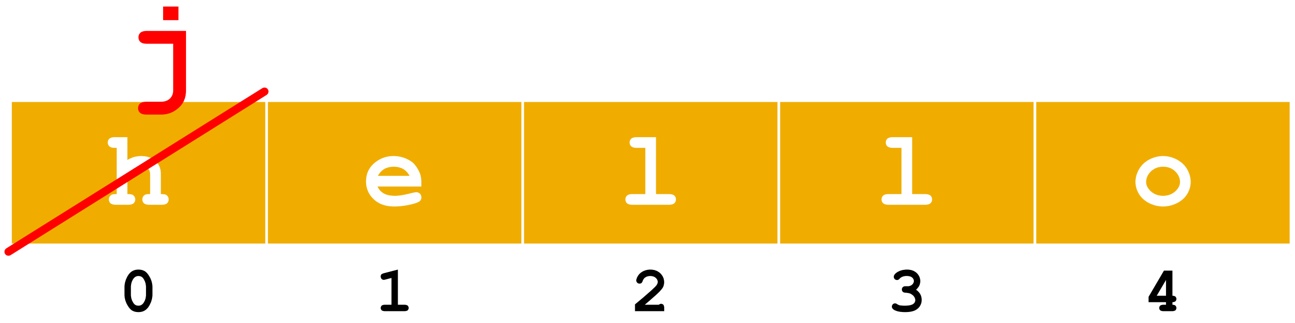
```
string str = "hello";
str[0] = 'j';
cout << str << endl;
```

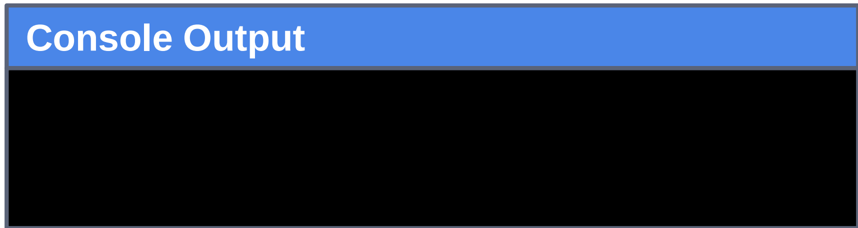


Console Output

chars can be accessed individually

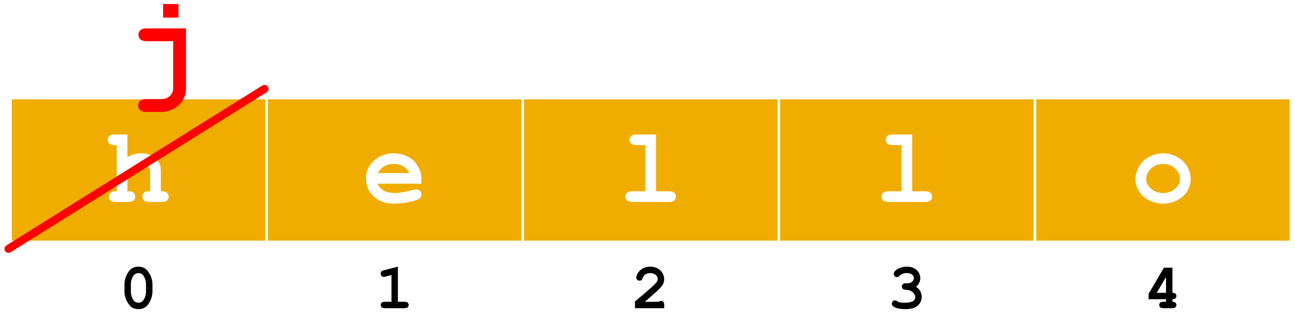
```
string str = "hello";
str[0] = 'j';
cout << str << endl;
```





chars can be accessed individually

```
string str = "hello";
str[0] = 'j';
cout << str << endl;
```

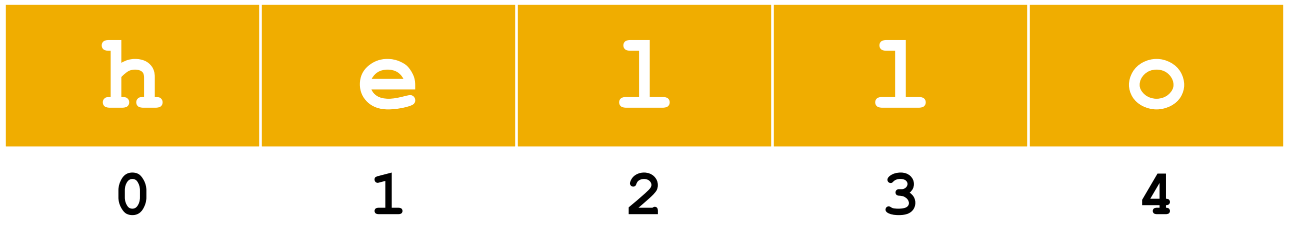


Console Output

```
jello
```

chars can be accessed individually

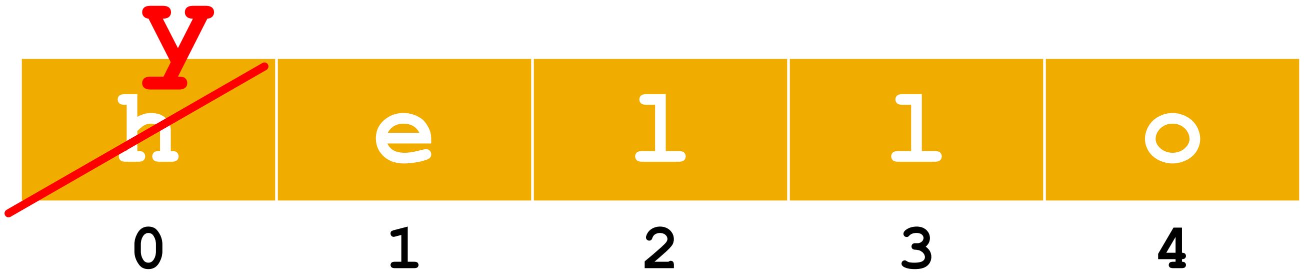
```
string str = "hello";
str[0] = 'y';
str[5] = 'w';
cout << str << endl;
```

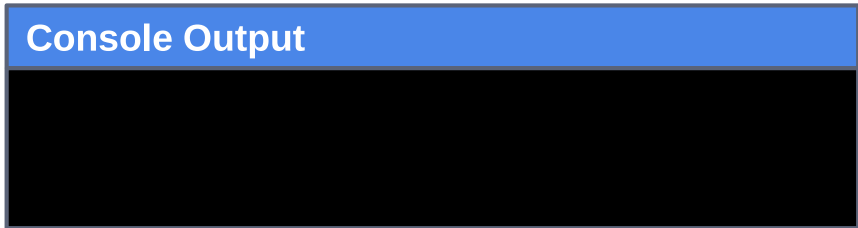


Console Output

chars can be accessed individually

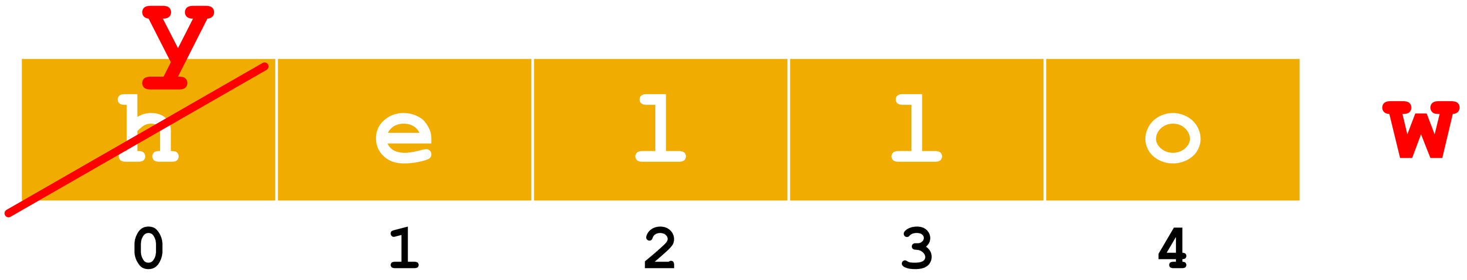
```
string str = "hello";
str[0] = 'y';
str[5] = 'w';
cout << str << endl;
```

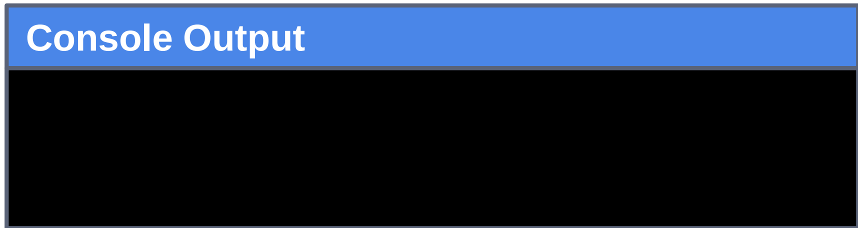




chars can be accessed individually

```
string str = "hello";
str[0] = 'y';
str[5] = 'w';
cout << str << endl;
```



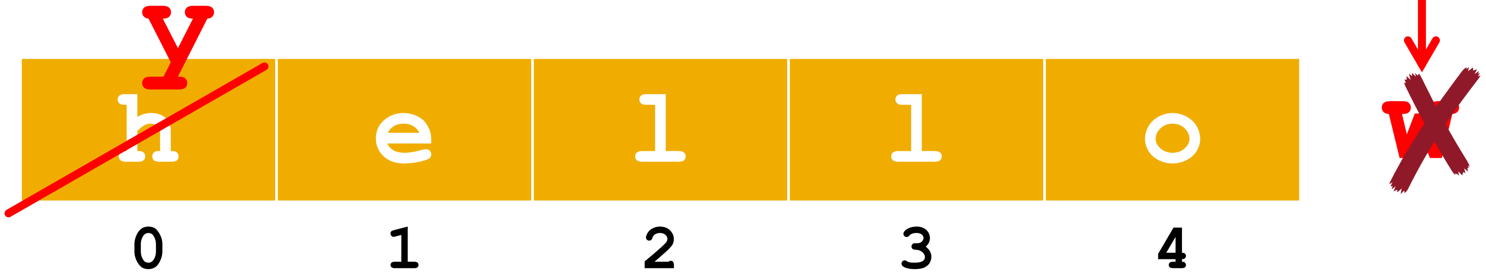


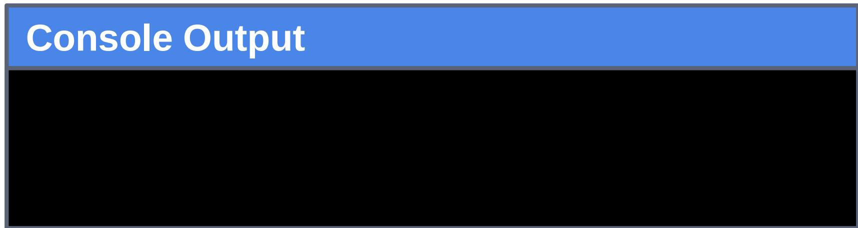
strings have a fixed length!

```
string str = "hello";
str[0] = 'y';
str[5] = 'w';
cout << str << endl;
```

RUNTIME ERROR!

Exceeds length of string!

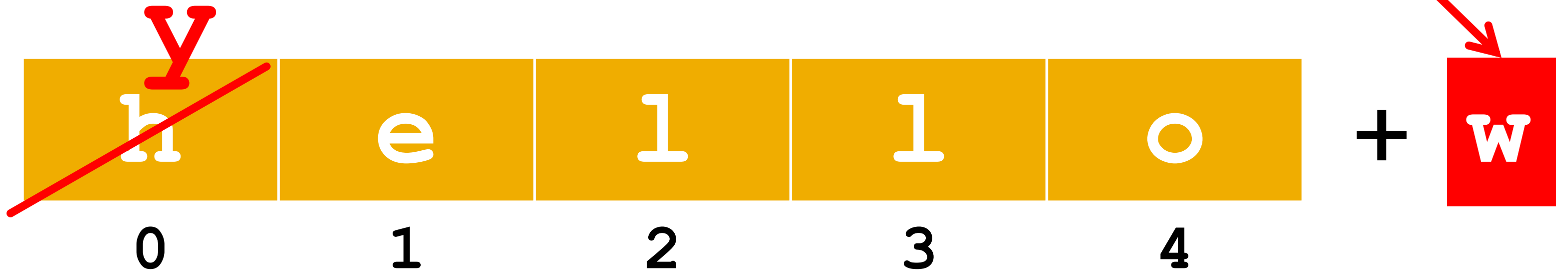




strings have a fixed length!

```
string str = "hello";
str[0] = 'y';
str = str + 'w';
cout << str << endl;
```

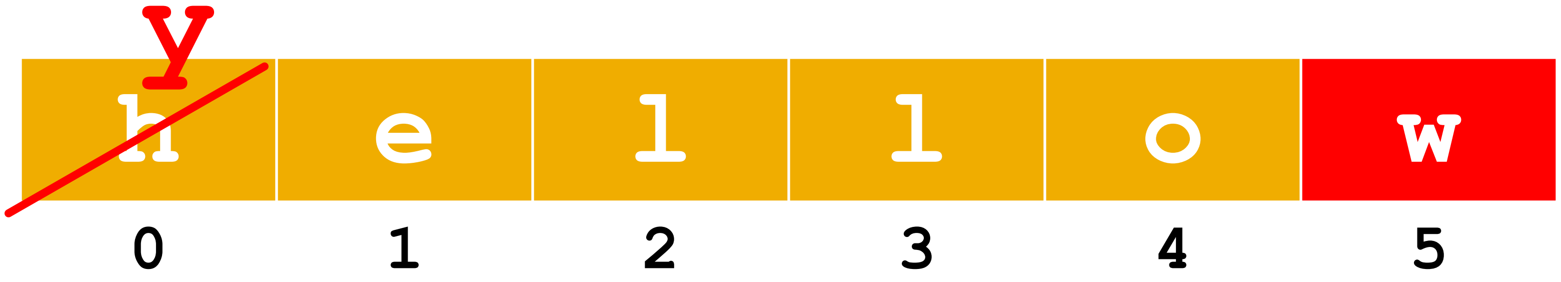
+ operator concatenates strings and characters

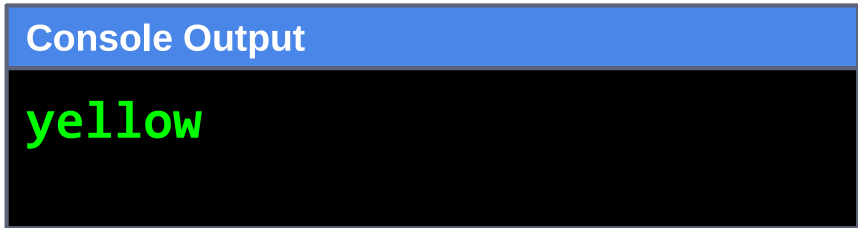


Console Output

strings have a fixed length!

```
string str = "hello";
str[0] = 'y';
str = str + 'w';
cout << str << endl;
```





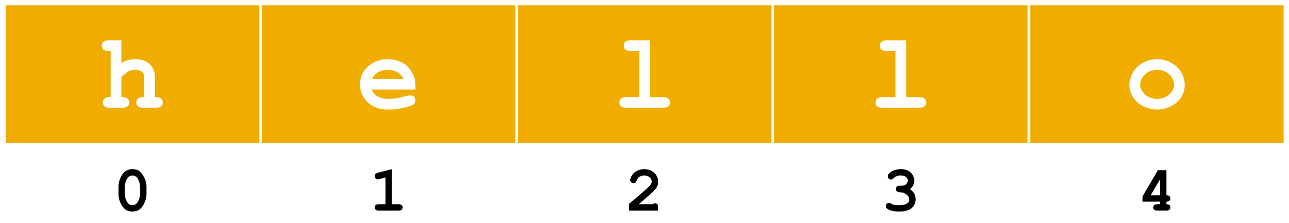
```
yellow
```

chars can be accessed individually

```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```

chars can be accessed individually

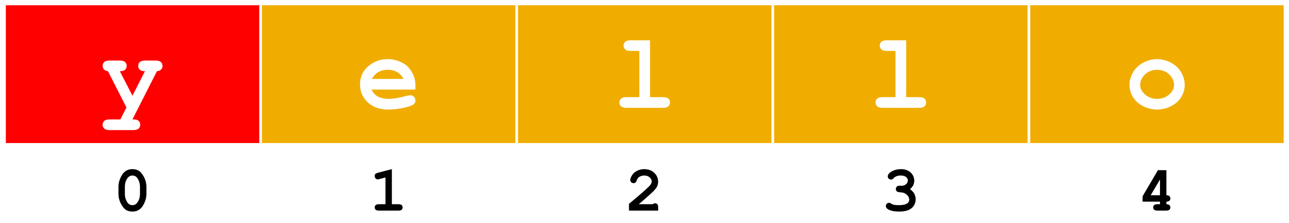
```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```

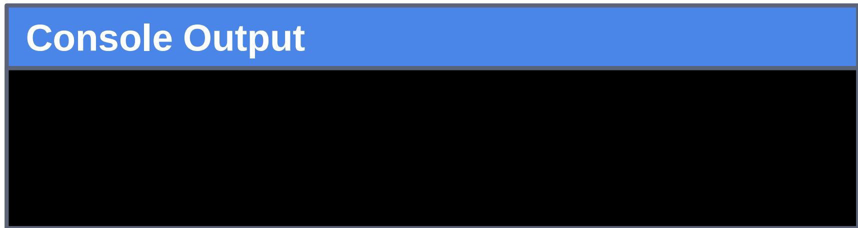


Console Output

chars can be accessed individually

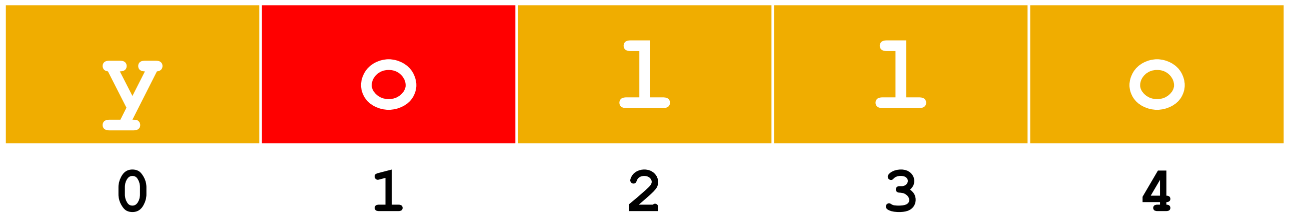
```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```

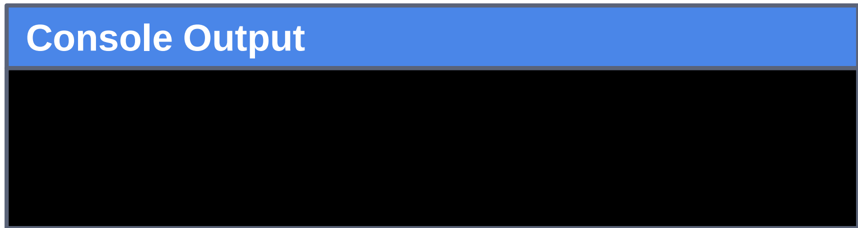




chars can be accessed individually

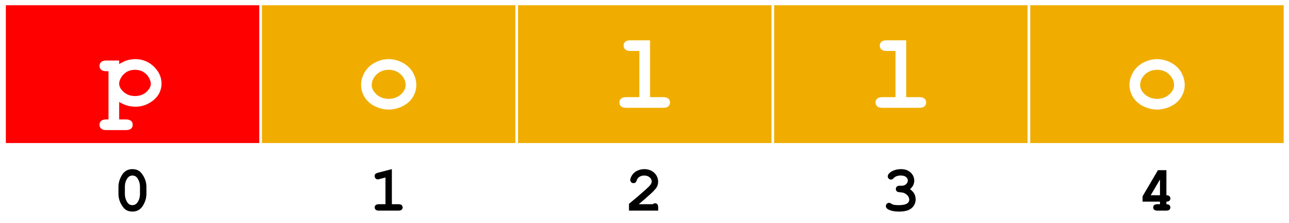
```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```

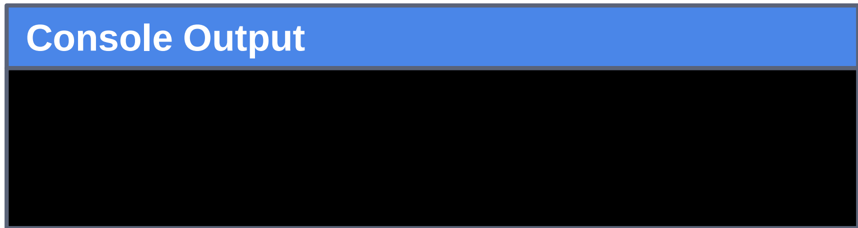




chars can be accessed individually

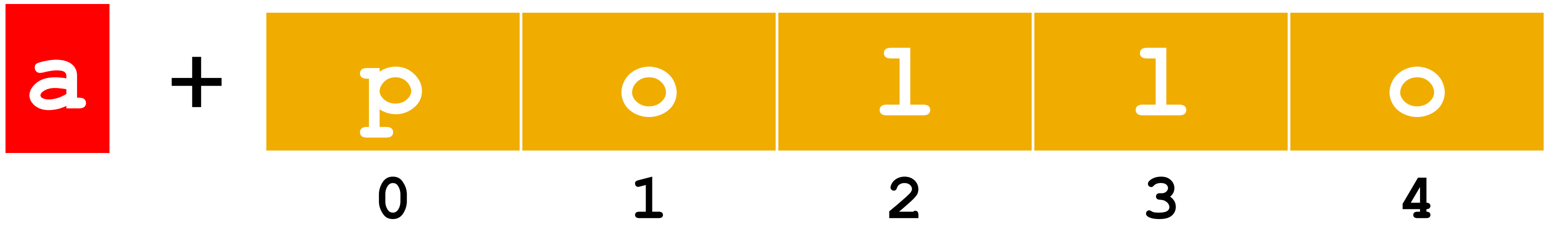
```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```

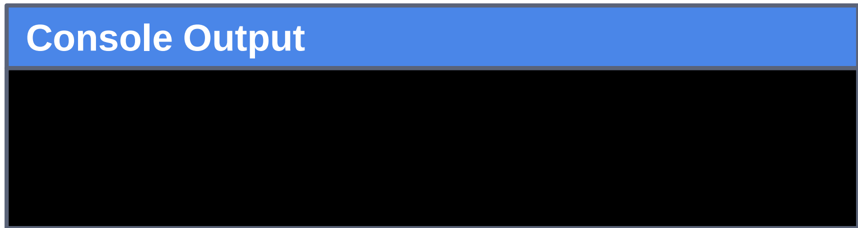




chars can be accessed individually

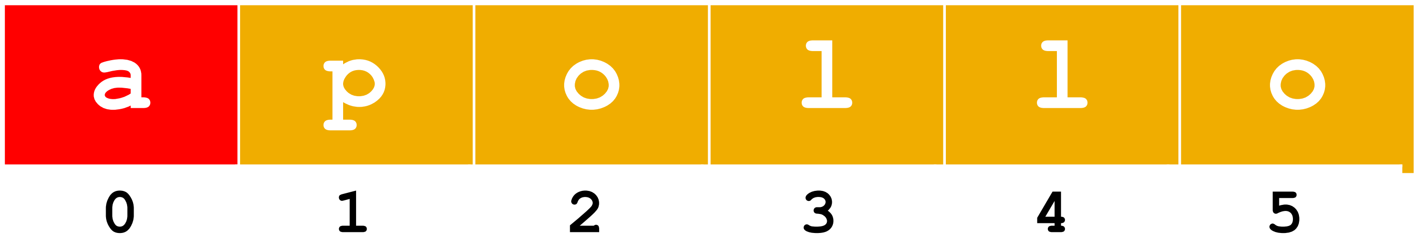
```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```





chars can be accessed individually

```
string str = "hello";
str[0] = 'y';
str[1] = 'o';
str[0] = 'p';
str = 'a' + str;
```



```
cout << str;
```

Console Output

```
apollo
```

Separate Compilation Header Files

Separate Source Files

Benefits of organizing code into separate functions:

- More readable
- More testable
- More reusable
- etc.

Same benefits for organizing groups of related functions and data into separate source files

We're Already Using Separate Source Files

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
    ...
}
```

Separate source files

We're Already Using Separate Source Files

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
    ...
}
```

Separate source files

Only declarations are required to use a function

We're Already Using Separate Source Files

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
    ...
}
```

Separate source files

Only declarations are required to use a function

Files consisting of just declarations are called *header* files

Example: statistics program

Normal .cpp file

- function declarations
- main function
- function definitions

stats.cpp

```
/**
 * Requires: variance >= 0.
 * Effects:  Computes the probability that a random sample
 *           from a normal distribution with the given mean
 *           and variance lies within the given range.
 */
double normalProbabilityInRange(double mean, double variance,
                                double low, double high);

int main() {

    cout << normalProbabilityInRange(0, 1, -1, 1);

}

double normalProbabilityInRange(double mean, double variance,
                                double low, double high) {
    return 0.0; // TODO: implement
}
```

Header file

- **function declarations**

stats.h

```
#ifndef STATS_H_
#define STATS_H_

// Statistical Functions

/**
 * Requires: variance >= 0.
 * Effects:  Computes the probability that a random sample
 *           from a normal distribution with the given mean
 *           and variance lies within the given range.
 */
double normalProbabilityInRange(double mean, double variance,
                                double low, double high);
```

stats.cpp

```
#include "stats.h"
#include <cmath>

// Statistical Functions

double normalProbabilityInRange(double mean, double variance,
                                double low, double high) {
    return 0.0; // TODO: implement
}
```

stats.h

```
#ifndef STATS_H_
#define STATS_H_

// Statistical Functions

/**
 * Requires: variance >= 0.
 * Effects:  Computes the probability that a random sample
 *           from a normal distribution with the given mean
 *           and variance lies within the given range.
 */
double normalProbabilityInRange(double mean, double variance,
                                double low, double high);
```

Header file

- function declarations

stats.cpp

```
#include "stats.h"
#include <cmath>

// Statistical Functions

double normalProbabilityInRange(double mean, double variance,
                                double low, double high) {
    return 0.0; // TODO: implement
}
```

cpp file

- function definitions

stats.h

```
#ifndef STATS_H_
#define STATS_H_

// Statistical Functions

/**
 * Requires: variance >= 0.
 * Effects:  Computes the probability that a random sample
 *           from a normal distribution with the given mean
 *           and variance lies within the given range.
 */
double normalProbabilityInRange(double mean, double variance,
                                double low, double high);
```

Header file

- function declarations

stats.cpp

```
#include "stats.h"
#include <cmath>

// Statistical Functions

double normalProbabilityInRange(double mean, double variance,
                                double low, double high) {
    return 0.0; // TODO: implement
}
```

cpp file

- function definitions

main.cpp file

- main function

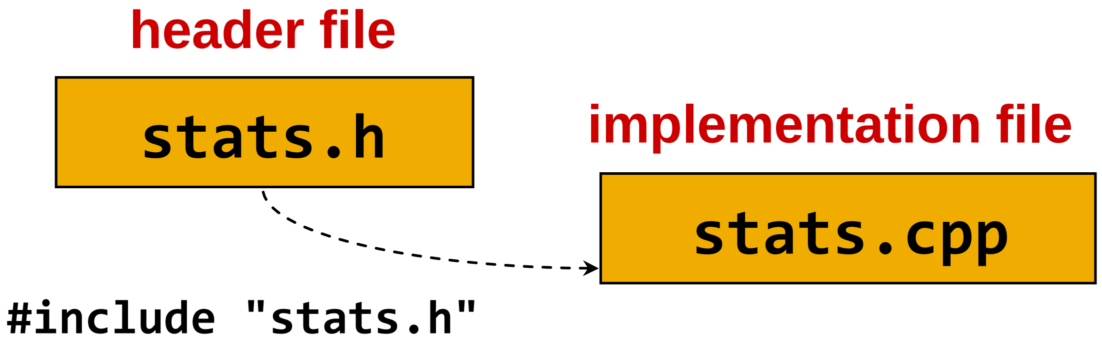
Separate Compilation and Linking of Files

header file

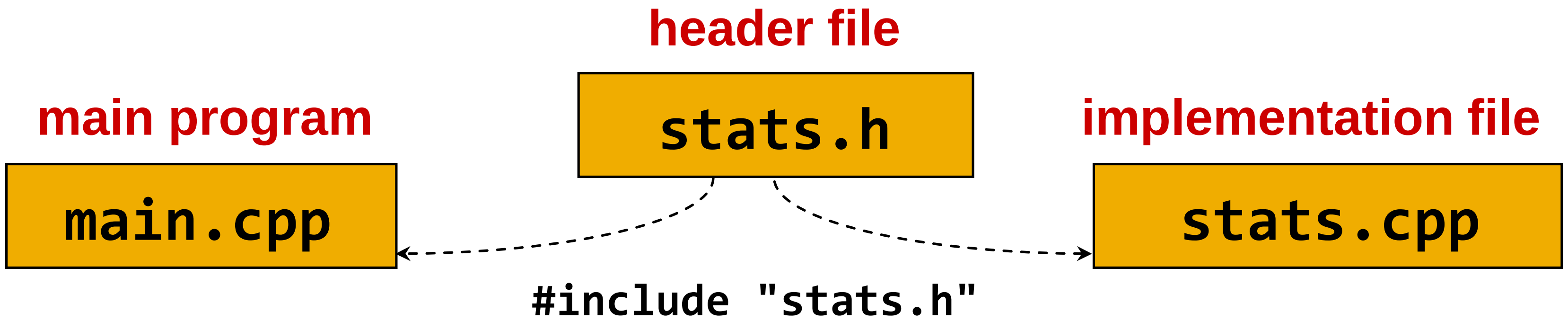
`stats.h`

```
#include "stats.h"
```

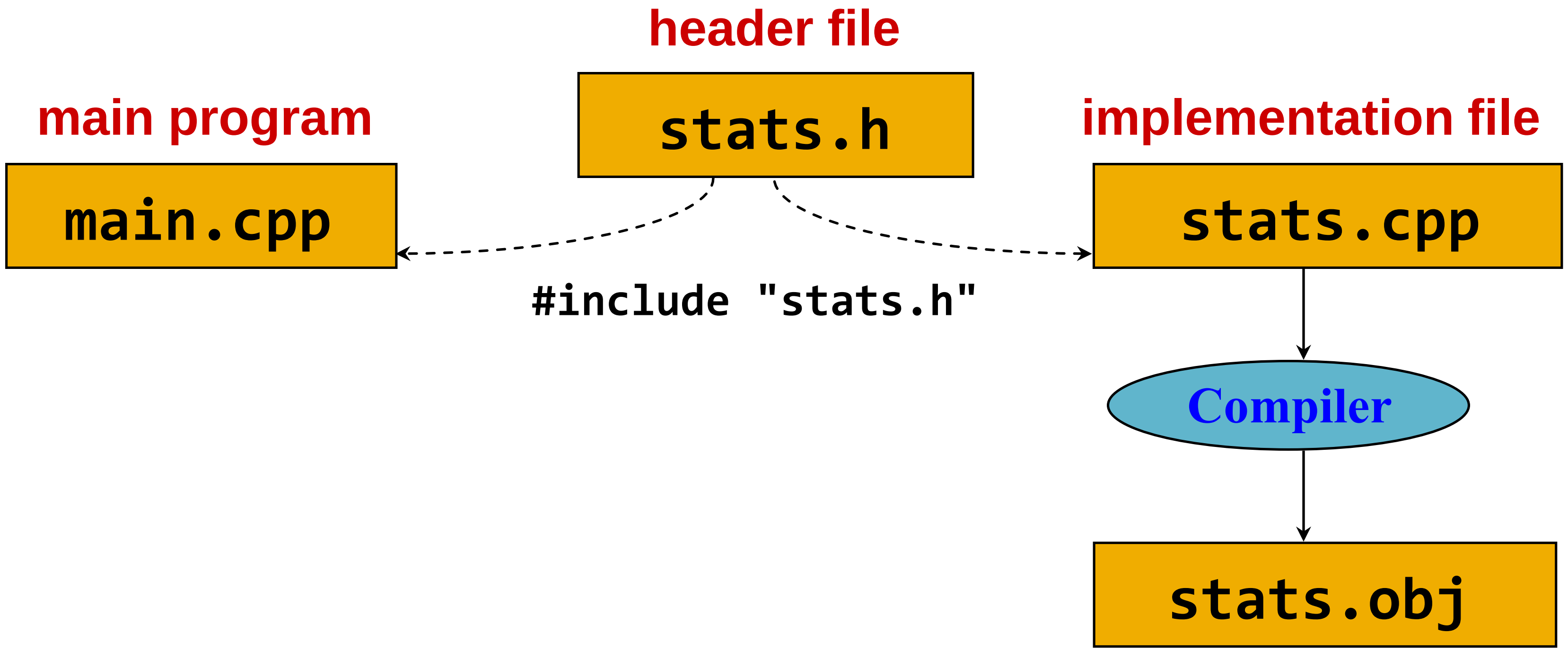
Separate Compilation and Linking of Files



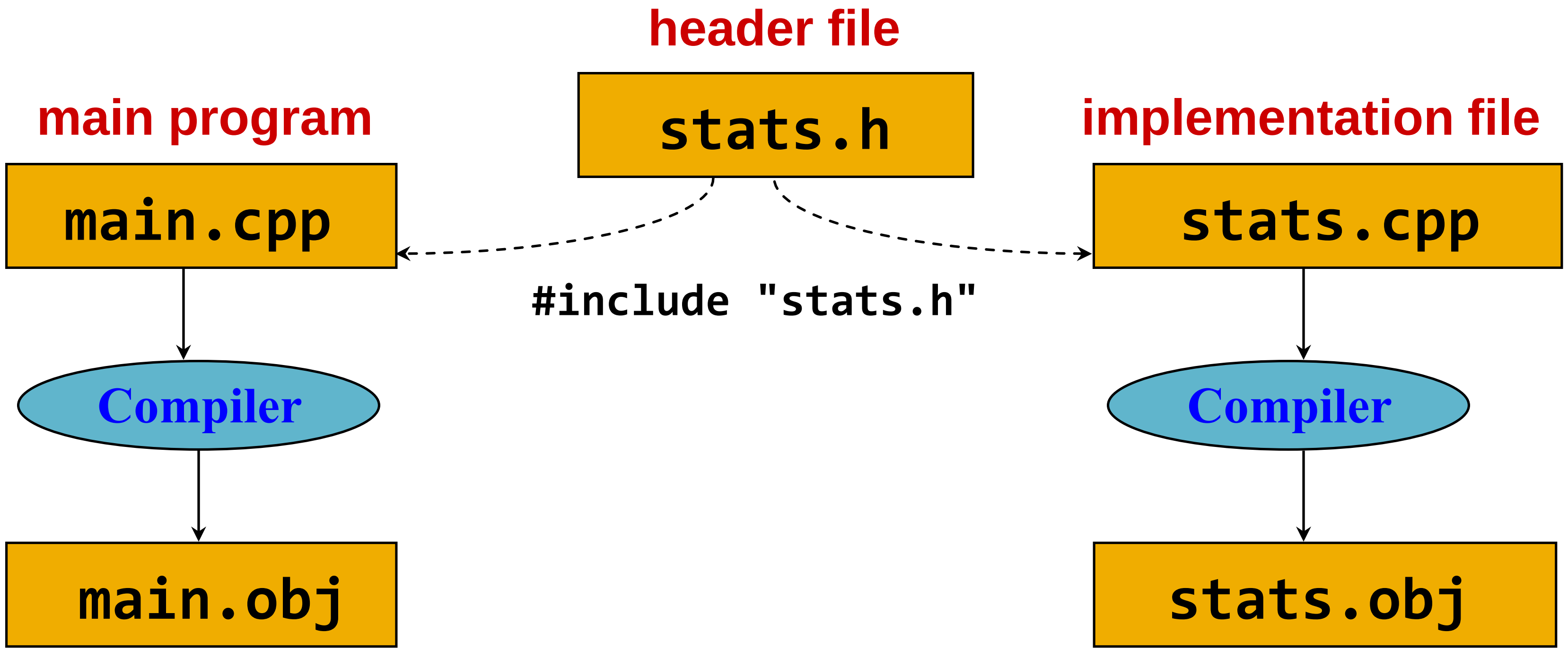
Separate Compilation and Linking of Files



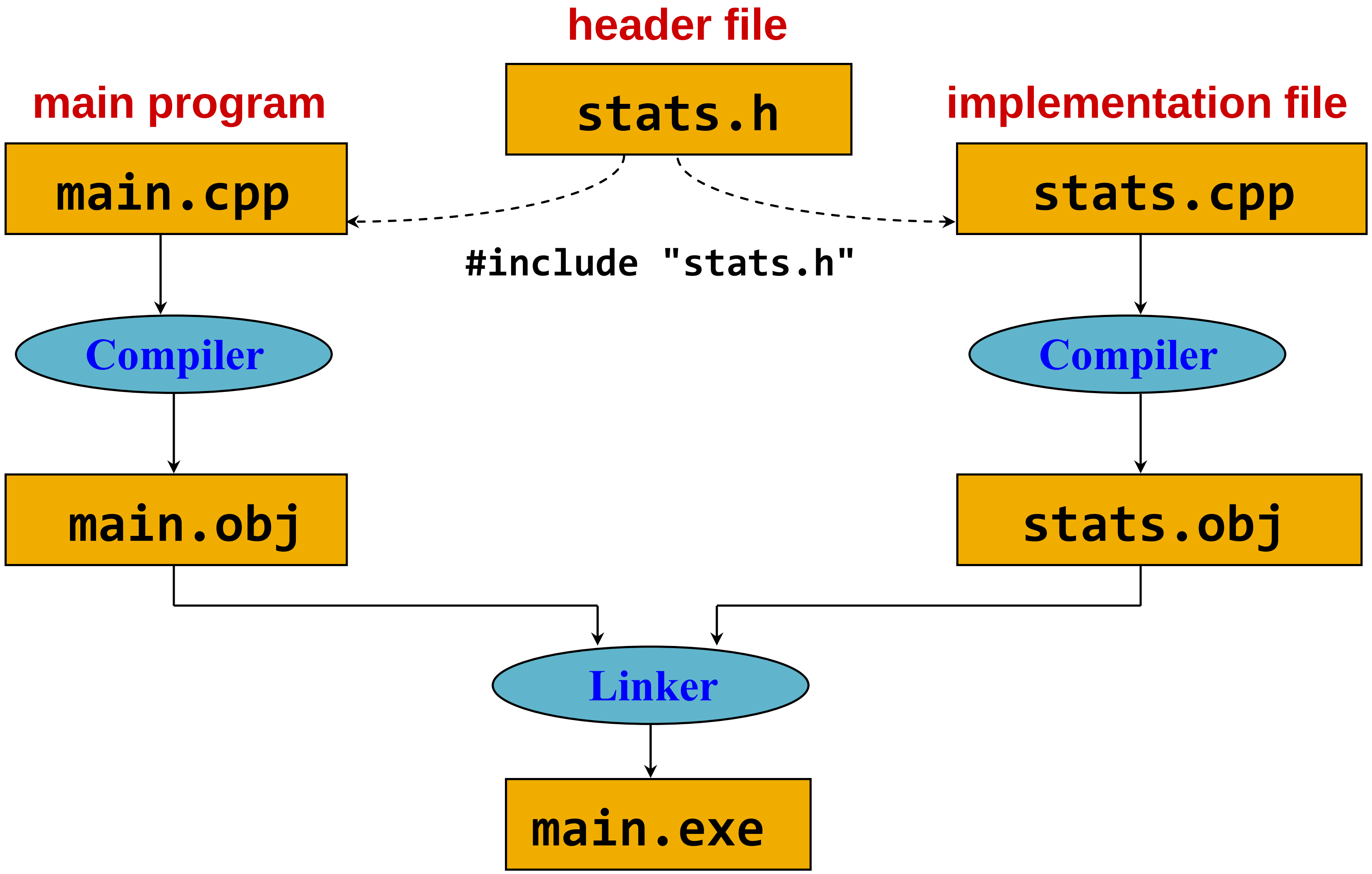
Separate Compilation and Linking of Files



Separate Compilation and Linking of Files



Separate Compilation and Linking of Files



That's all folks

On your own:

See following slides for examples

Write as a “`for`” loop

```
i = 10;
while (i > 0) {
   cout << 10 / i << endl;
   i--;
}
```

Write as a "`for`" loop

```
i = 10;
while (i > 0) {
   cout << 10 / i << endl;
   i--;
}
```

```
for (i = 10; i > 0; i--) {
   cout << 10 / i << endl;
}
```

How many times does the following code print ‘x’?

```
int i = 5;
while (i)
{
   cout << "x";
   i = i - 2;
}
```

How many times does the following code print ‘x’?

```
int i = 5;
while (i)
{
   cout << "x";
   i = i - 2;
}
```

infinite

Any non-0 value is true!

What prints?

```
int i = 0, j = 0;
while (j < 5)
{
   int count = 0;
   while (i < 5)
   {
      count++;
   }
}
cout << count;
```

a) code won’t compile
b) 0
c) 5
d) 25
e) nothing – infinite loop

What prints?

```
int i = 0, j = 0;
while (j < 5)
{
   int count = 0;
   while (i < 5)
   {
      count++;
   }
}
cout << count;
```

a) **code won’t compile**
b) 0
c) 5
d) 25
e) nothing – infinite loop

What prints?

```
int i = 0, j = 0;
while (j < 5)
{
   int count = 0;
   while (i < 5)
   {
      count++;
   }
}
cout << count;
```

a) **code won't compile**
b) 0
c) 5
d) 25
e) nothing – infinite loop

What prints?

```
int j;
for (int i = 1; i < 4; i++)
{
   j = 1;
   while (i >= j)
   {
      cout << '*';
      j++;
   }
   cout << endl;
}
```

What prints?

```
int j;
for (int i = 1; i < 4; i++)
{
   j = 1;
   while (i >= j)
   {
      cout << '*';
      j++;
   }
   cout << endl;
}
```

```
*
**
***
```

What prints?

```
int j = 4, k = 10;
while (j < k)
{
   j++;
   k-=2;
}
cout << j << " " << k;
```

What prints?

```
int j = 4, k = 10;
while (j < k)
{
   j++;
   k-=2;
}
cout << j << " " << k;
```

j	k
4	10
5	8
6	6

```
6 6
```

which has correct syntax

A) `for (int i = 0; i < 5; i++)`

B) `for (; ; )`

C) `for (; i > 5 && i <= 10; i--)`

D)
```
for (int j = 10, k = 0;
     j > 0 && k < 20;
     j--, k++)
```

E) all of the above

which has correct syntax

```
A) for (int i = 0; i < 5; i++)
B) for (; ; )
C) for (; i > 5 && i <= 10; i--)
D) for (int j = 10, k = 0;
        j > 0 && k < 20;
        j--, k++)
E) all of the above
```

Sum of first n positive integers

```
// Requires: n > 0
// Effects: Returns sum of first n integers
// That is, returns 1 + 2 + … + n
int sum(int n)
{
   int s = 0;

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

   return s;
}
```

Sum of first n positive integers

```
// Requires: n > 0
// Effects: Returns sum of first n integers
// That is, returns n + … + 2 + 1
int sum(int n)
{
   int s = 0;

   for ( int i = n; i > 0; i-- )
   {
      s = s + i;
   }


   return s;
}
```

Prime

```
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
  for ( int i = 2; i < n; i++ )
  {
    if ( n % i == 0 )
    {
       return false;
    }
  }
  return true;
}
```

What prints?

```
int n = 3;

for (int i = 10; i <= n; i++) {
   cout << i << " ";
 }
```

A. 1 2 3
B. 10 11 12
C. Nothing
D. Error

Loop body may execute zero times

```
  int n = 3;

  for (int i = 10; i <= n; i++) {
     cout << i << endl;
}
```

loop body never executes

A. 1 2 3
B. 10 11 12
C. Nothing
D. Error

What prints?

```
const int MAX =  9;
int row;
for (row = 1; row <= MAX; row++) {
    cout << row << ' ';
}
```

A) 1 2 3 4 5 6 7 8
B) 1 2 3 4 5 6 7 8 9
C) Neither of the above

What prints?

Pattern: 1 to N loop

```
const int MAX =  9;
int row;
for (row = 1; row <= MAX; row++) {
     cout << row << ' ';
}
```

A) 1 2 3 4 5 6 7 8
B) 1 2 3 4 5 6 7 8 9
C) Neither of the above

What prints?

```
const int TOTAL = 10;
int i = 4, sum = 0;

for (i; i < TOTAL; i++)
{
  if ((TOTAL / (sum + 1)) == 1)
      sum += 3;
  else
      sum += 1;
}
cout << sum << " " << i;
```

A. sum = 3, i = 1
B. sum = 9, i = 9
C. sum = 10, i = 10
D. sum = 12, i = 10
E. sum = 8, i = 10

What prints?

```
const int TOTAL = 10;
int i = 4, sum = 0;

for (i; i < TOTAL; i++)
{
  if ((TOTAL / (sum + 1)) == 1)
      sum += 3;
  else
      sum += 1;
}
cout << sum << " " << i;
```

```
 sum    i
----- ---
  0     4
  1     5
  2     6
  3     7
  4     8
  5     9
  8    10
```

What prints?

```
const int TOTAL = 10;
int i = 4, sum = 0;

for (i; i < TOTAL; i++)
{
  if ((TOTAL / (sum + 1)) == 1)
      sum += 3;
  else
      sum += 1;
}
cout << sum << " " << i;
```

A. sum = 3, i = 1
B. sum = 9, i = 9
C. sum = 10, i = 10
D. sum = 12, i = 10
E. sum = 8, i = 10

What prints?

```
int limit = 8;
cout << 'H';

for ( int i = 10; i <= limit; i++)
{
    cout << 'E';
}
cout << "LP";
```

A)HLP
B)HELP
C)HEELP
D)HEEELP
E)none of the above

What prints?

```
int limit = 8;
cout << 'H';

for ( int i = 10; i <= limit; i++)
{
    cout << 'E';
}
cout << "LP";
```

never enters loop

A)HLP
B)HELP
C)HEELP
D)HEEELP
E)none of the above

What prints?

```
int total = 0;
int num = 3;

for (int i = 0; i < num; i++)
{
    total += i;
}

cout << total << endl;
```

A) 0
B) 2
C) 3
D) none of the above

What prints?

```
int total = 0;
int num = 3;

for (int i = 0; i < num; i++)
{
   total += i;
}

cout << total << endl;
```

Pattern: 0 to N-1 loop

A) 0
B) 2
C) 3
D) none of the above

What prints?

```
int total = 0;
int num = 3;

for (int i = 0; i < num; i++)
{
   total += i;
}

cout << i << endl;
```

A) 0
B) 2
C) 3
D) none of the above

What prints?

```
int total = 0;
int num = 3;

for (int i = 0; i < num; i++)
{
    total += i;
}
```

Scope of i

```
cout << i << endl;
```

i is out of scope

A) 0
B) 2
C) 3
D) none of the above

What prints?

```
int total = 0;
int num = 3;
int i;

for (i = 0; i < num; i++)
{
    total += i;
}

cout << i << endl;
```

A) 0
B) 2
C) 3
D) none of the above

What prints?

```
int total = 0;
int num = 3;
int i;

for (i = 0; i < num; i++)
{
    total += i;
}

cout << i << endl;
```

Scope of i

A) 0
B) 2
C) 3
D) none of the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 0; i < num; i++) {
    total *= i;
}
```

```
cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 0; i < num; i++) {
    total *= i;
}
```

```
cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 1; i < num; i++) {
    total *= i;
}
```

```
cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 1; i < num; i++) {
    total *= i;
}

cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of
   the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 1; i <= num; i++) {
    total *= i;
}
```

```
cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 1; i <= num; i++) {
    total *= i;
}
```

```
cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 1; i <= num; i++) {
    total *= num;
}
```

```
cout << total << endl;
```

A) 0
B) 2
C) 6
D) none of
   the above

What prints?

```
int total = 1;
int num = 3;

for ( int i = 1; i <= num; i++) {
    total *= num;
}
```

cout << total << endl;

Output:

A) 0
B) 2
C) 6
D) none of
the above

What prints?

```
for ( int j = 0; j < 3; j++) {
    cout << '*';
}
```

A) *
B) **
C) ***
D) none of the above

What prints?

```
for ( int j = 0; j < 3; j++) {
    cout << '*';
}
```

A) *

B) **

C) ***

D) none of the above