Loops (Continued)

- do-while loops
- for loops
What prints?

```cpp
int i = 0;

while (i < 5 )
{
    cout << i << endl;
    i++;
}
```
What prints?

```cpp
int i = 0;
while (i < 5) {
    cout << i << endl;
    i++;
}
```
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!
The 'do-while' loop

- Concept is the same as the while loop, except:
  - Test is at **end** of the loop
  - Executed **at least once**
The do-while loop...

Syntax

do {
    statements;
} while (condition);
The do-while loop...

Syntax

do {
    statements;
} while (condition);
I notice you're doing a while loop there. I'd do a do-while loop, personally.

While you do do-while, I do while.

You do? Do you do do-while while I do while?

No, I do not do do-while while you do while. Nor do I do do-while while you do do-while. I do not do do-while at all.

Do you do do-while while you do nested loops?

Why do you not do do-while while whiling?

Well, while you wile away and not do do-while I'll do do-while.

No, I do not do do-while while whiling.

While can do what do-while can do, while do-while can't do what while can do.

Do.
What prints?

int i = 0;

while (i < 5 )
{
    cout << i << endl;
    i++;
}
int i = 0;

while (i < 5 )
{
    cout << i << endl;
    i++;
}
What prints?

```cpp
int i = 0;

do
{
    cout << i << endl;
    i++;
} while (i < 5);
```
int i = 0;

do {
    cout << i << endl;
    i++;
} while (i < 5);
If a loop can be executed zero times you can use:
If a loop can be executed *zero* times you can use:
- `while` loop
If a loop can be executed zero times you can use:

- while loop
- do-while loop
Notes

- If a loop can be executed **zero** times you can use:
  - `while` loop
  - `do-while` loop

- If a loop must be executed **at least once** you can use:
If a loop can be executed zero times you can use:
- while loop
- do-while loop

If a loop must be executed at least once you can use:
- while loop
Notes

- If a loop can be executed **zero** times you can use:
  - `while` loop
  - `do-while` loop

- If a loop must be executed **at least once** you can use:
  - `while` loop
  - `do-while` loop (preferable)
Sum of first n positive integers

```c
int sum(int n)
{
    int sum = 0;
    int i = 1;
    while (i <= n)
    {
        sum = sum + i;
        i = i + 1;
    }
    return sum;
}
```
Sum of first n positive integers

// Requires:
// Effects:

int sum(int n)
{

}
// Requires: n > 0
// Effects:

int sum(int n)
{
    int sum = 0;
    int i = 1;
    while (i <= n)
    {
        sum = sum + i;
        i = i + 1;
    }
    return sum;
}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{

}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
//          That is, returns 1 + 2 + ... + n
int sum(int n)
{
    int sum = 0;
    int i = 1;
    while (i <= n)
    {
        sum = sum + i;
        i = i + 1;
    }
    return sum;
}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{

    while ( )
    {

    }
}
/ *Requires: n > 0*

// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n

```c
int sum(int n)
{

    int i = 1;
    while ( (i <= n) )
    {

        i = i + 1;
    }
}
```
// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{
    int s = 0;
    int i = 1;
    while ( i <= n )
    {
        s = s + i;
        i = i + 1;
    }
}

// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{
    int s = 0;
    int i = 1;
    while ( i <= n )
    {
        s = s + i;
        i = i + 1;
    }
    return s;
}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{
    int s = 0;
    int i = 1;
    while ( i <= n )
    {
        s = s + i;
        i = i + 1;
    }
    return s;
}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
//          That is, returns n + ... + 2 + 1
int sum(int n)
{

    while ( )
    {
        
    }
}

// Requires: n > 0
// Effects: Returns the sum of the first n integers
//          That is, returns n + ... + 2 + 1
int sum(int n)
{

    int i = n;
    while ( i > 0 )
    {

        i = i - 1;
    }

}
Sum: With loop counting backwards

// Requires: n > 0
// Effects: Returns the sum of the first n integers
//          That is, returns n + ... + 2 + 1
int sum(int n)
{
    int s = 0;
    int i = n;
    while ( i > 0 )
    {
        s = s + i;
        i = i - 1;
    }
    return s;
}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
//          That is, returns n + ... + 2 + 1
int sum(int n)
{
    int s = 0;
    int i = n;
    while (i > 0)
    {
        s = s + i;
        i = i - 1;
    }
    return s;
}
I have nothing to do, so I'm trying to calculate the prime factors of the time each minute before it changes.

It was easy when I started at 1:00, but with each hour the number gets bigger.

I wonder how long I can keep up.

253 is $11 \times 23$

WHAT?

I'm factoring the time.

Hey!

Think fast.
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n) {

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

bool prime(int n) {
    int i = 2;
    while (i < n) {
        if (n % i == 0) {
            return false;
        }
        i = i + 1;
    }
    return true;
}
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
    int i = 2;
    while (i < n)
    {
        i = i + 1;
    }
}
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
    int i = 2;
    while (i < n)
    {
        if (n % i == 0)
        {
            i = i + 1;
        }
    }
    return true;
}
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
    int i = 2;
    while (i < n)
    {
        if (n % i == 0)
        {
            return false;
        }
        i = i + 1;
    }
    return true;
}
// Requires: \( n > 1 \)
// Effects: Returns true iff \( n \) is prime

```cpp
bool prime(int n)
{
    int i = 2;
    while ( i < n )
    {
        if ( n % i == 0 )
        {
            return false;
        }
        i = i + 1;
    }
    return true;
}
```
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
    int i = 2;
    while ( i < n )
    {
        if ( n % i == 0 )
        {
            return false;
        }
        i = i + 1;
    }
    return true;
}
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
    int i = 2;               // initialization
    while (i < n)           // check
    {
        if (n % i == 0)
        {
            return false;
        }
        i = i + 1;           // update
    }
    return true;
}
while (count < 10) {
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}

4 16
5 25
6 36
7 49
8 64
9 81
Squares

count = 4; // initialization
while (count < 10) // check
{
    square = count * count;
    cout << count << " " << square << endl;
    count++; // update
}

4 16
5 25
6 36
7 49
8 64
9 81
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}

4   16
5   25
6   36
7   49
8   64
9   81
Squares

```cpp
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}
```

4 16
5 25
6 36
7 49
8 64
9 81
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}

(count = 4; count < 10; count++)
{
}

Squares
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}

(count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
}
while loop vs for loop

```cpp
int count = 4;
while (count < 10)
{
    int square = count * count;
    cout << count << " " << square << endl;
    count++;
}
```

```cpp
for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << " " << square << endl;
}
```
for loop

- Another general iteration construct
- Typical usage: count-controlled loops
- Allows convenient specification of initialization, check, & update
for loop : SYNTAX

for ( expr1; expr2; expr3 )
{
    statements;
}

for loop: SYNTAX

Initialization
Check
Update

for ( expr1; expr2; expr3 )
{
    statements;
}

Loop body
for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << " " << square << endl;
}

initialization
```c++
for (int count = 4; count < 10; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}
```
Example: Squares

for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << "  " << square << endl;
}

loop body
for (int count = 4; count < 10; count++) {
    int square = count * count;
    cout << count << "  " << square << endl;
}

for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << " " << square << endl;
}
for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << "  " << square << endl;
}
Example: Squares

```cpp
for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << "  " << square << endl;
}
```
for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << "  " << square << endl;
}

check
Example: Squares

```cpp
for (int count = 4; count < 10; count++)
{
    int square = count * count;
    cout << count << " " << square << endl;
}
```

check
continue until condition is false
for (count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}
for (count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;  
}
What if we increment counter within loop body

```cpp
for (count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++; 
}

while ( )
{
}
```
What if we increment counter within loop body

```cpp
for (count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;  
}
```

```cpp
count = 4;
while (count < 10)
{
    count++;
}  
```
What if we increment counter within loop body

```cpp
for (count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}
```

```cpp
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
    count++;
}
```
What if we increment counter within loop body

```cpp
for (count = 4; count < 10; count++)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
}
```

```cpp
count = 4;
while (count < 10)
{
    square = count * count;
    cout << count << " " << square << endl;
    count++;
    count++;
}
```
Initialization

for (<exp1>; <exp2>; <exp3>) <statement>

- Can be omitted

for ( ; <expr2>; <expr3>) {...}
Initialization

```plaintext
for (<exp1>; <exp2>; <exp3>) <statement>
```

- Can be omitted
  ```plaintext
  for (; <expr2>; <expr3>) {...}
  ```

- Can use *comma expression* to initialize more than one variable
  ```plaintext
  for (count = 1, sum = 0; <expr2>; <expr3>) {...}
  ```
Initialization

- Can be omitted

```cpp
for ( ; <expr2>; <expr3>) {...}
```

- Can use *comma expression* to initialize more than one variable

```cpp
for (count = 1, sum = 0; <expr2>; <expr3>) {...}
```

- Can declare variable (or multiple variables of the same type)

```cpp
for (int i = 1; <expr2>; <expr3>) {...}
```
Initialization

for (<exp1>; <exp2>; <exp3>) <statement>

- Can be omitted

  for ( ; <expr2>; <expr3>) {...}

- Can use *comma expression* to initialize more than one variable

  for (count = 1, sum = 0; <expr2>; <expr3>) {...}

- Can declare variable (or multiple variables of the same type)

  for (int i = 1; <expr2>; <expr3>) {...}

  *i* is only visible inside the **for** loop
Initialization

for (<exp1>; <exp2>; <exp3>) <statement>

- Can be omitted

  for ( ; <expr2>; <expr3>) {…}

- Can use *comma expression* to initialize more than one variable

  for (count = 1, sum = 0; <expr2>; <expr3>) {…}

- Can declare variable (or multiple variables of the same type)

  for (int i = 1; <expr2>; <expr3>) {…}
  for (int i = 1, j = 1; <expr2>; <expr3>) {…}
Update

```plaintext
for (<expr1>; <exp2>; <exp3>) <statement>
```

- Can be omitted

```plaintext
for (<expr1>; <expr2>; ) {...}
```
for (<expr1>; <expr2>; <exp3>) <statement>

- Can be omitted
  
  for (<expr1>; <expr2>; ) {...}

- Can use *comma expression* to update more than one variable

  for (<expr1>; <expr2>; i++, j++) {...}
Check

for (<expr1>; <exp2>; <exp3>) <statement>

- Can be omitted (in which case the condition evaluates to true)

  for (<expr1>; ; <expr3>) {...}
for (<expr1>; <expr2>; <expr3>) <statement>

- Can be a single simple statement

  for (<expr1>; <expr2>; <expr3>) s+=i;
for (<expr1>; <expr2>; <expr3>) <statement>

- Can be a **single** simple statement
  
  for (<expr1>; <expr2>; <expr3>) \( s += i; \)

- Can be a **single** compound statement (containing multiple statements)
  
  for (<expr1>; <expr2>; <expr3>) {int t=x; x=y; y=t;}
  for (<expr1>; <expr2>; <expr3>) if (b) x = y; else y = x;
for (<expr1>; <exp2>; <exp3>) <statement>

- Can be a single simple statement

  for (<expr1>; <exp2>; <exp3>) s+=i;

- Can be a single compound statement (containing multiple statements)

  for (<expr1>; <exp2>; <exp3>) {int t=x; x=y; y=t;}
  for (<expr1>; <exp2>; <exp3>) if (b) x = y; else y = x;

- Can be a single empty statement

  for (<expr1>; <exp2>; <exp3>) ;
  for (<expr1>; <exp2>; <exp3>) {}
Same applies to body of while

**while (<condition>) <statement>**

- Can be a **single** simple statement
  
  ```c
  while (<condition>) s+=i;
  ```

- Can be a **single** compound statement (containing multiple statements)
  
  ```c
  while (<condition>) {
    int t=x; x=y; y=t;
  }
  while (<condition>) if (b) x = y; else y = x;
  ```

- Can be a **single** empty statement
  
  ```c
  while (<condition>) ;
  while (<condition>) {}
  ```
if (<condition>) <statement>

- Can be a single simple statement
  
  if (<condition>) s+=i;

- Can be a single compound statement (containing multiple statements)
  
  if (<condition>) {int t=x; x=y; y=t;}
  if (<condition>) if (b) x = y; else y = x;

- Can be a single empty statement
  
  if (<condition>) ;
  if (<condition>) {}
Same applies to body of else

if (<cond>)<statement1> else <statement2>

- Can be a single simple statement

```
if (<condition>) <statement1> else s+=i;
```

- Can be a single compound statement (containing multiple statements)

```
if (<condition>) <statement1> else {int t=x; x=y; y=t;}
if (<condition>) <statement1> else if (b) x = y; else y = x;
```

- Can be a single empty statement

```
if (<condition>) <statement1> else ;
if (<condition>) <statement1> else {}
```
Sum of first n positive integers

// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{

}
// Requires: n > 0
// Effects: Returns the sum of the first n integers
//          That is, returns 1 + 2 + ... + n
int sum(int n)
{

    for ( ; ; )
    {

    }

}
Sum of first n positive integers

// Requires: n > 0
// Effects: Returns the sum of the first n integers
// That is, returns 1 + 2 + ... + n
int sum(int n)
{

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

}
// Requires: n > 0
// Effects: Returns the sum of the 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;
    }

}
// Requires: n > 0
// Effects: Returns the sum of the 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;
}
// Requires: n > 0
// Effects: Returns the sum of the 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 the sum of the 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;
    }

    cout << i;

    return s;
}
// Requires: n > 0
// Effects: Returns the sum of the 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;
    }  

    // cout << i; // i is out of scope here

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

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

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

    return s;
}
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{

}
/\ 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;
}
// Requires: n > 1
// Effects: Returns true iff n is prime

bool prime(int n)
{
    for ( int i = 2; i < n; i++ )
    {
    }
}

// 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;
}
/\ 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;
}
// 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;
}
for (char ch = 'a'; ch <= 'z'; ch++)
{
    cout << ch;
}
for (char ch = 'a'; ch <= 'z'; ch++)
{
    cout "<< ch;
}

prints (assuming the ASCII character set):

abcdef...wxyz
int n = 3;

for (i = 10; i <= n; i++) {
    cout << i << endl;
}
int n = 3;

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

loop body never executes
const int MAX = 9;

int row;

for (row = 1; row <= MAX; row++) {
    cout << row << endl;
}

What prints

```cpp
const int MAX = 9;

int row;

for (row = 1; row <= MAX; row++) {
    cout << row << endl;
}
```
const int TOTAL = 10;
int i = 4, sum = 0;

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

cout << sum << " " << i;
int limit = 8;

cout << 'H';

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

cout << "LP";
int total = 0;
int n = 3;

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

cout << total << endl;
What prints

```cpp
int total = 0;
int n = 3;

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

cout << total << endl;
cout << i << endl;
```
What prints

```cpp
int total = 0;
int n     = 3;

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

cout << total << endl;
cout << i << endl;  // i is out of scope
```
int total = 0;
int n = 3;
int i;
for ( i = 0; i < n; i++)
{
    total += i;
}

cout << total << endl;
cout << i << endl;
int total = 0;
int n = 3;

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

cout << total << endl;
int total = 0;
int n = 3;

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

cout << total << endl;
int total = 1;
int n = 3;

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

cout << total << endl;

//out of scope
int total = 1;
int n     = 3;

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

cout << total << endl;

//out of scope
int total = 1;
int n = 3;

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

cout << total << endl;
Write as a “for” loop

\[
i = 10;
\]
\[
\text{while } (i > 0) \{
\]
\[
\quad \text{cout} \ll 10 \div i \ll \text{ endl} ;
\]
\[
\quad i--;\]
\[
\}
\]
Write as a “for” loop

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

```cpp
for (i = 10;
    i > 0;
    i--)
{
    cout << 10 / i << endl;
}
```
How many times does the following code print ‘x’?

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

infinite
What prints?

for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 3; j++)
    {
        cout << "*";
    }
    cout << endl;
}
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 3; j++)
    {
        cout << '*';
    }
    cout << endl;
}

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

What prints?

a) code won’t compile
b) 0
c) 5
d) 25
e) nothing – infinite loop
int i = 0, j = 0;
while (j < 5)
{
    int count = 0;
    while (i < 5)
    {
        count++;
    }
}
cout << count;
What prints?

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

*  
**  
***
int j = 4, k = 10;
while (j < k)
{
    j++;
    k-=2;
}
cout << j << " " << k;
int j = 4, k = 10;
while (j < k)
{
    j++;
    k-=2;
}
cout << j << " " << k;
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
Cin >> date;

While (date < 31)

Is it the 31st yet?

Nope..!
date++;
Late Days

- 3 late days per semester
- Use on any project
- Just submit
- We pull last submits, and calculate them

"A virus ate my homework."
Late Days

- Late Days used posted in gradebook
  - (NOT in autograder – only a guideline)

- 4\(^{th}\) Late Day
  - 50% deduction

- 5\(^{th}\) Late Day
  - score of 0