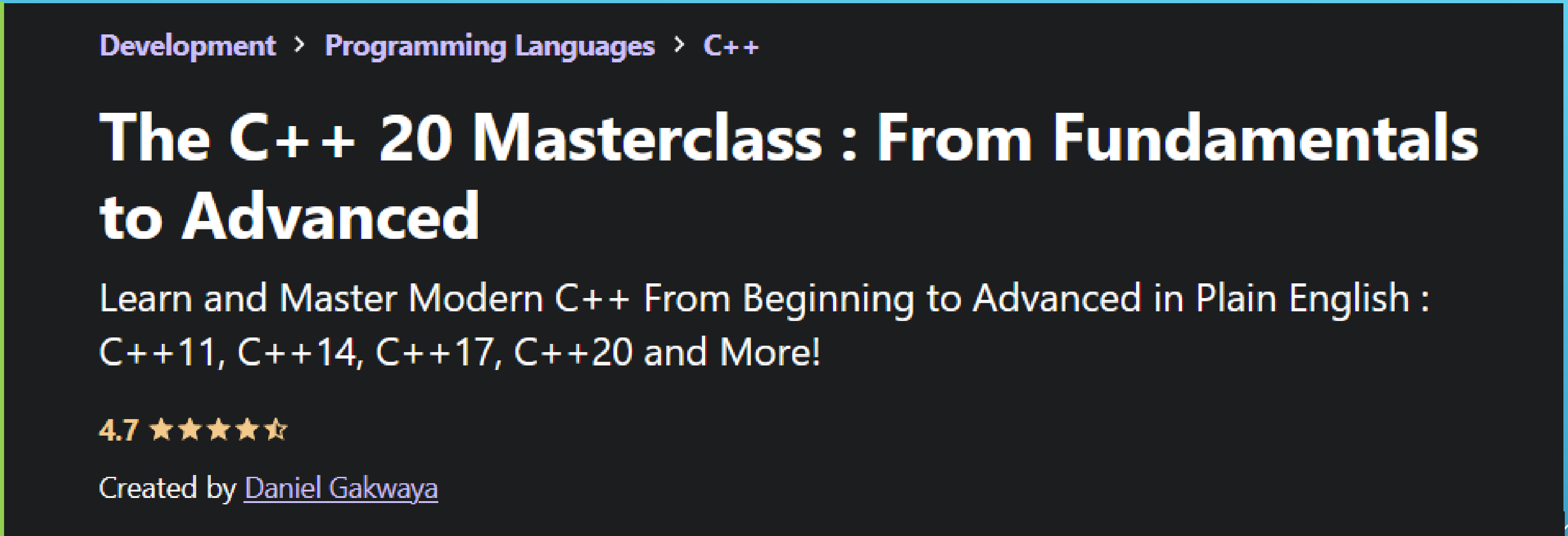
Slides



Development › Programming Languages › C++

The C++ 20 Masterclass : From Fundamentals to Advanced

Learn and Master Modern C++ From Beginning to Advanced in Plain English : C++11, C++14, C++17, C++20 and More!

4.7 ★★★★⯪

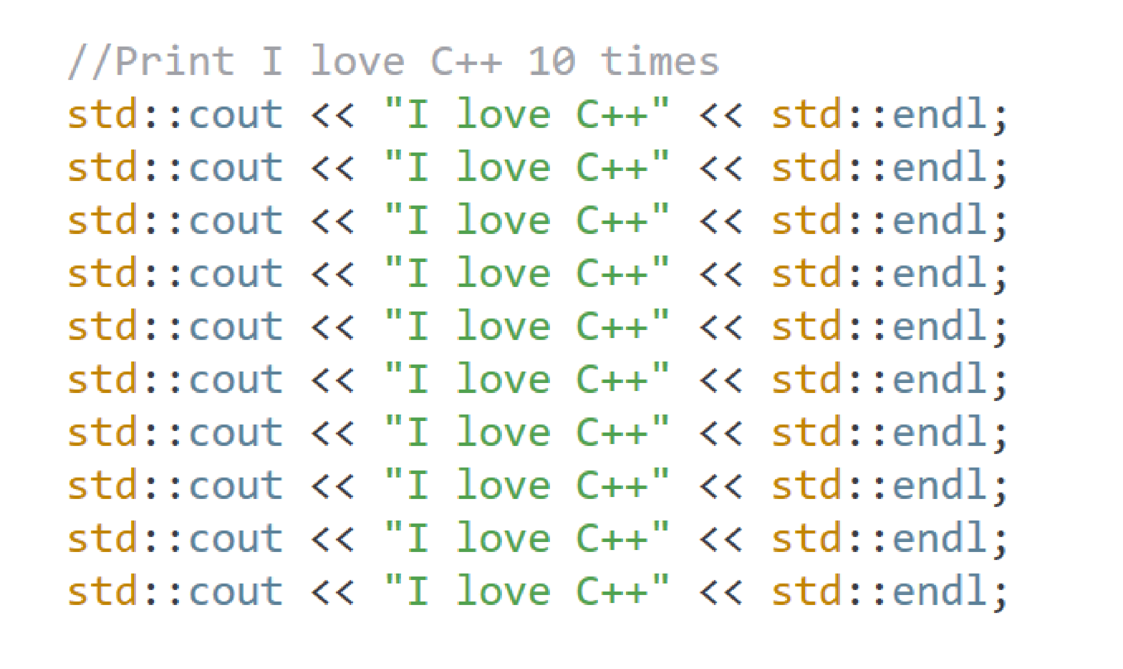
Created by Daniel Gakwaya

Section : Loops

Slide intentionally left empty

Loops : Introduction

Repetitive tasks



```
//Print I love C++ 10 times
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
```

For loop

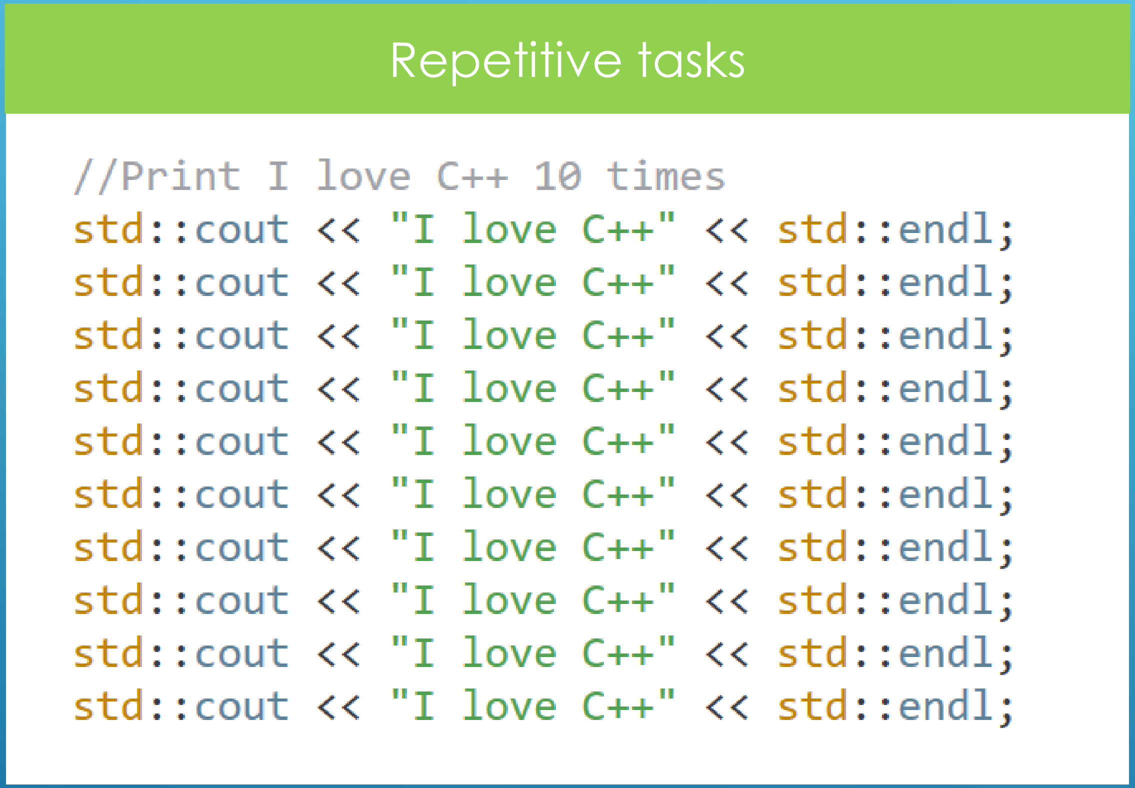
Range based for loop

While loop

Do while loop

Slide intentionally left empty

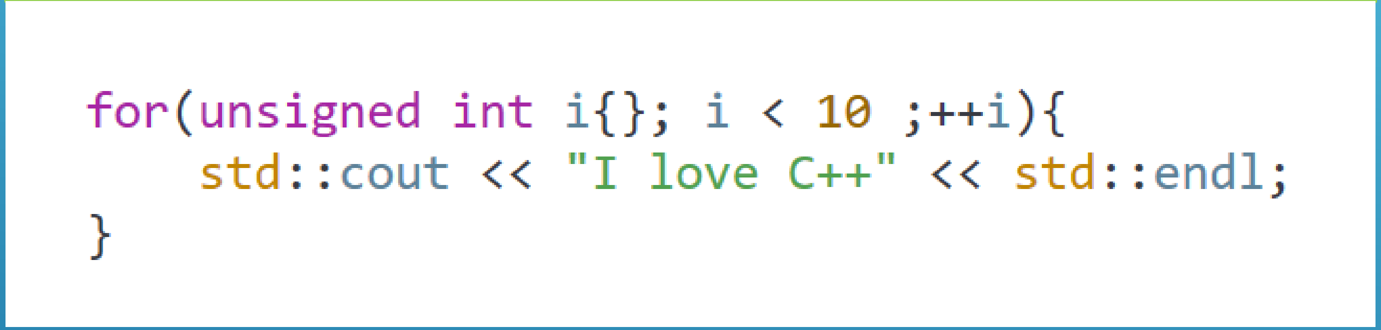
for loop



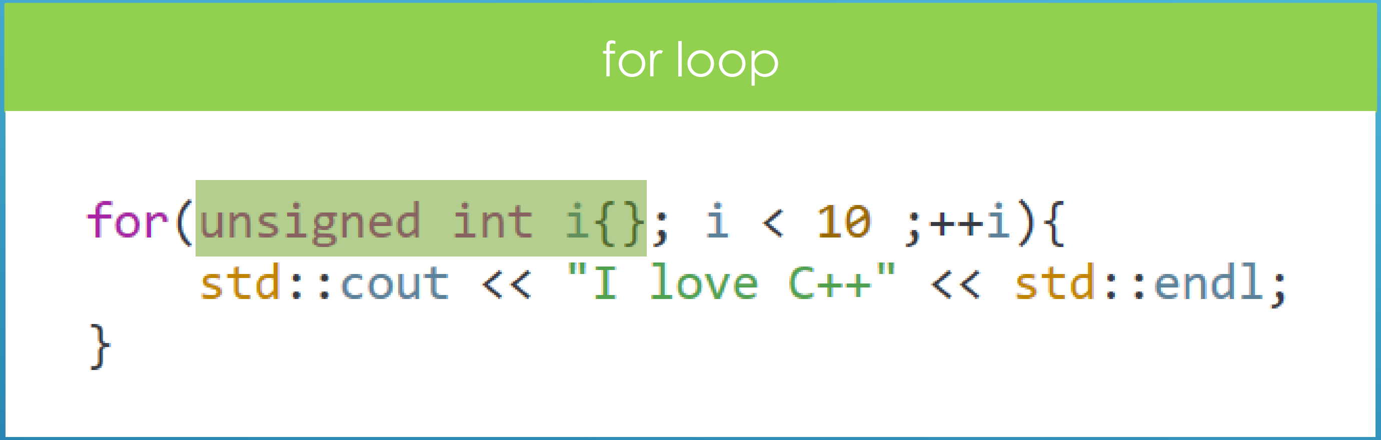
Repetitive tasks

```cpp
//Print I love C++ 10 times
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
```

for loop



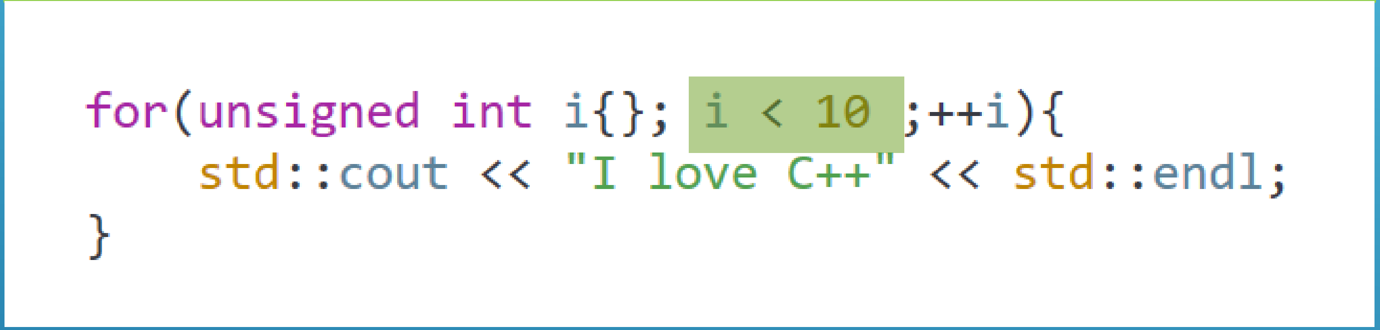
```cpp
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```



for loop

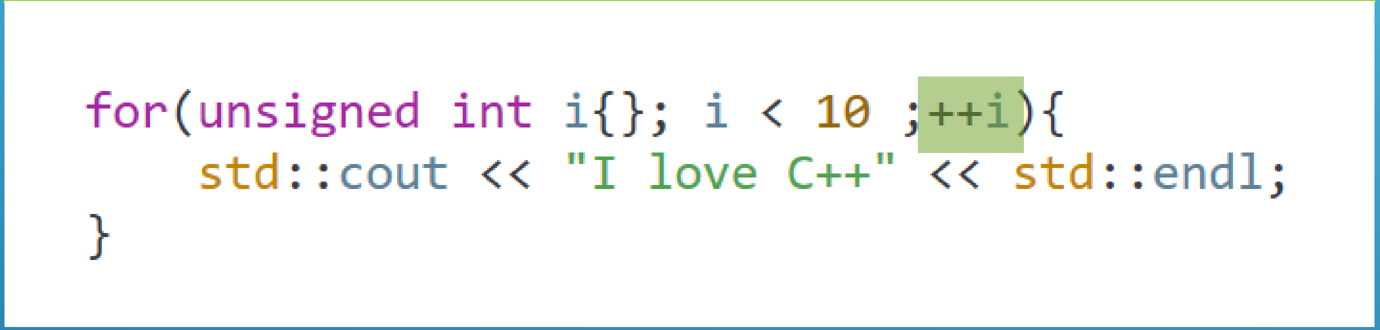
```cpp
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

for loop



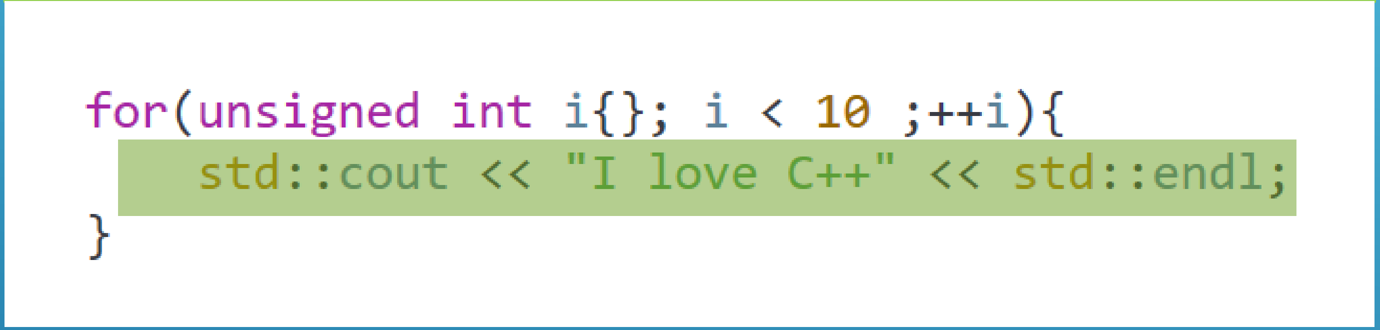
```
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

for loop



```
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

for loop



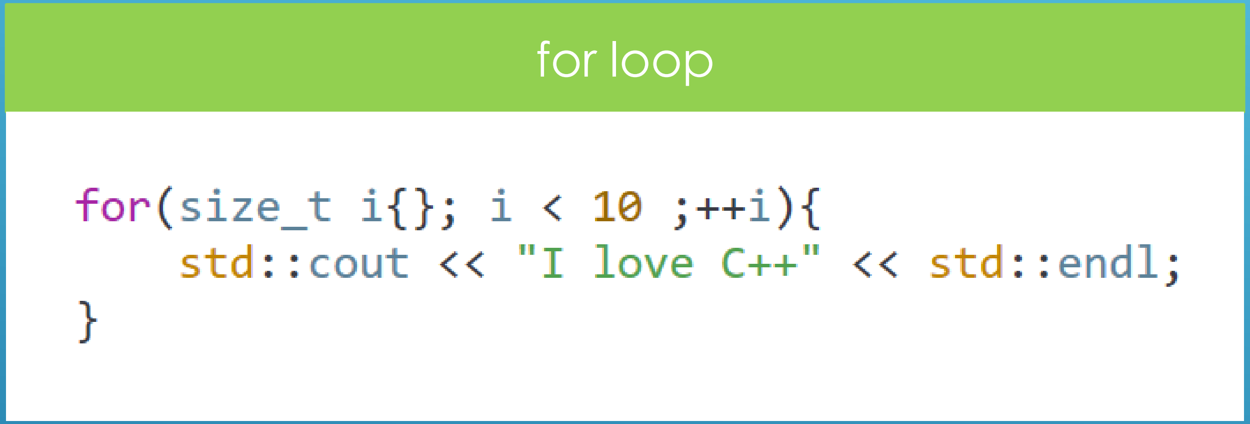
```
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

Pillars of any loop

- Iterator
- Starting Point
- Test(controls when the loop stops)
- Increment(Decrement)
- Loop body

size_t

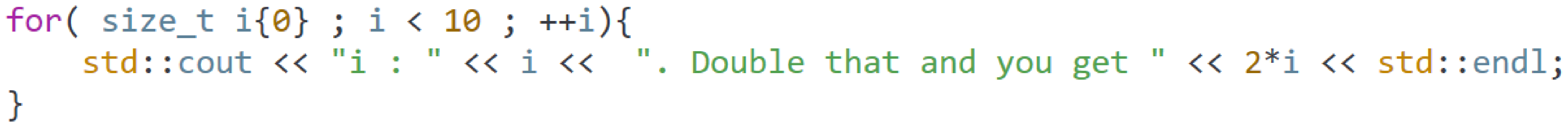
Not a type, just a type alias for some unsigned int representation



for loop

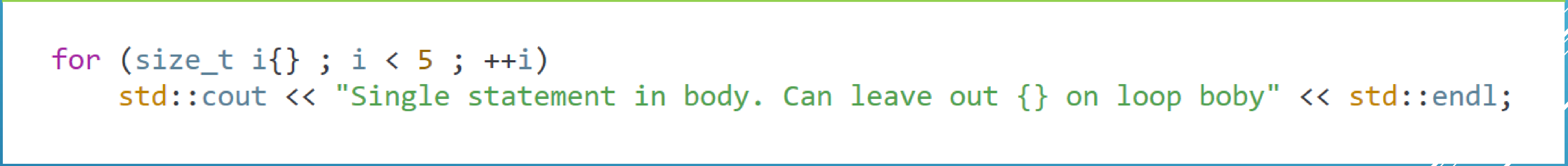
```cpp
for(size_t i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

Other operations in the loop body



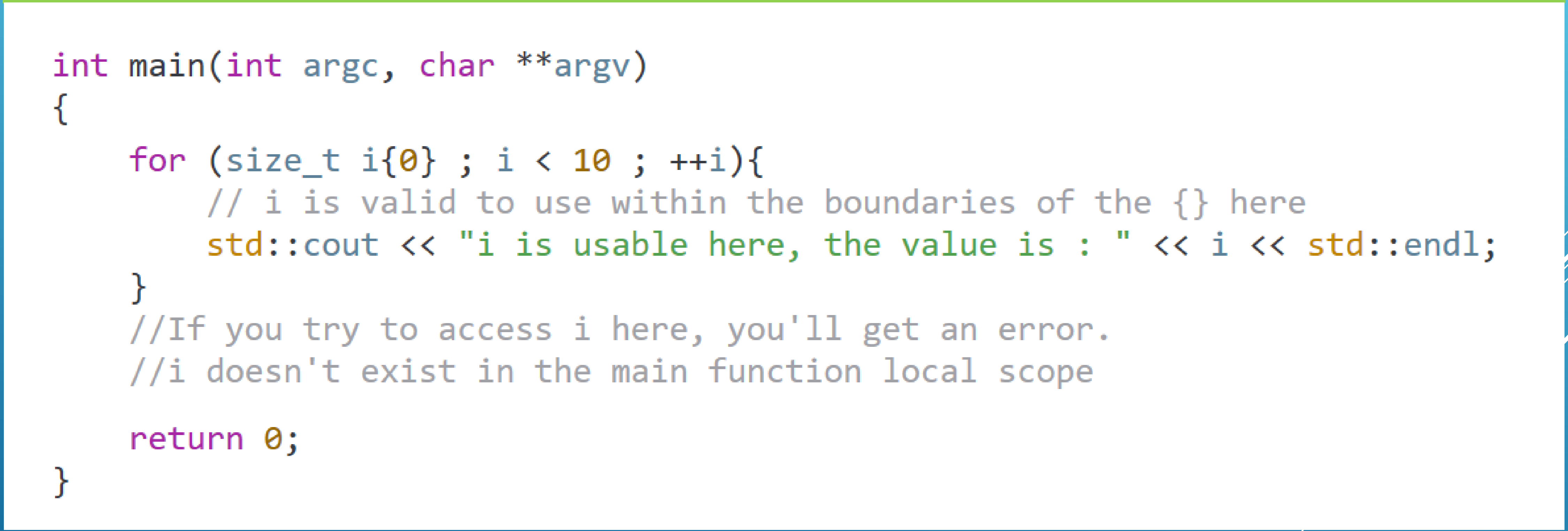
```
for( size_t i{0} ; i < 10 ; ++i){
    std::cout << "i : " << i <<  ". Double that and you get " << 2*i << std::endl;
}
```

Can leave out the {}



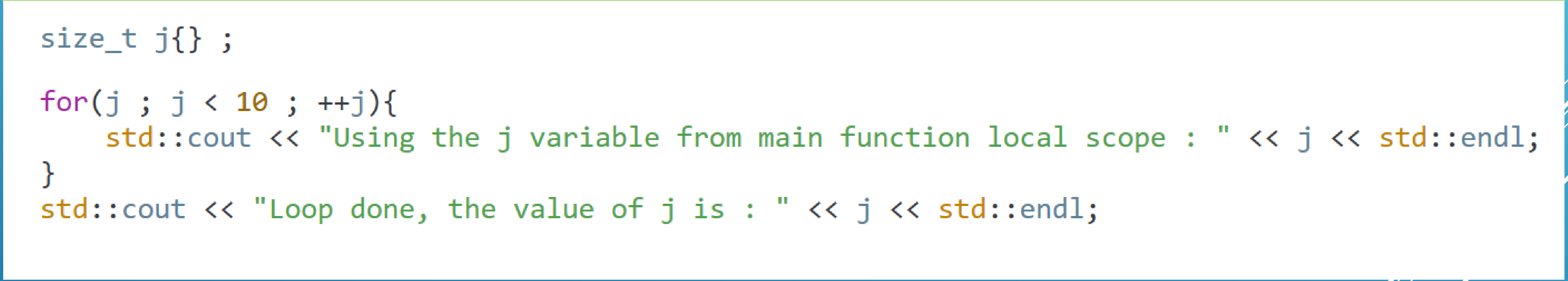
```
for (size_t i{} ; i < 5 ; ++i)
    std::cout << "Single statement in body. Can leave out {} on loop boby" << std::endl;
```

Scope of the iterator



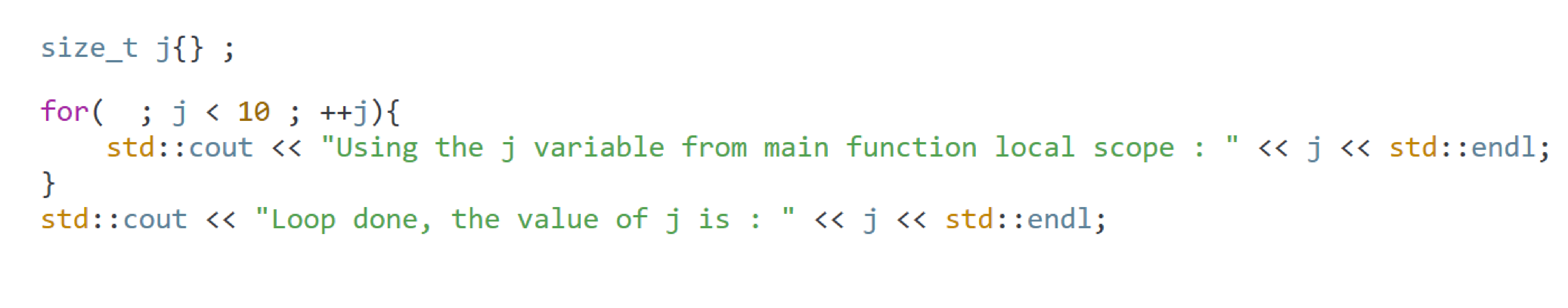
```cpp
int main(int argc, char **argv)
{
    for (size_t i{0} ; i < 10 ; ++i){
        // i is valid to use within the boundaries of the {} here
        std::cout << "i is usable here, the value is : " << i << std::endl;
    }
    //If you try to access i here, you'll get an error.
    //i doesn't exist in the main function local scope

    return 0;
}
```

Iterator can live outside the loop scope



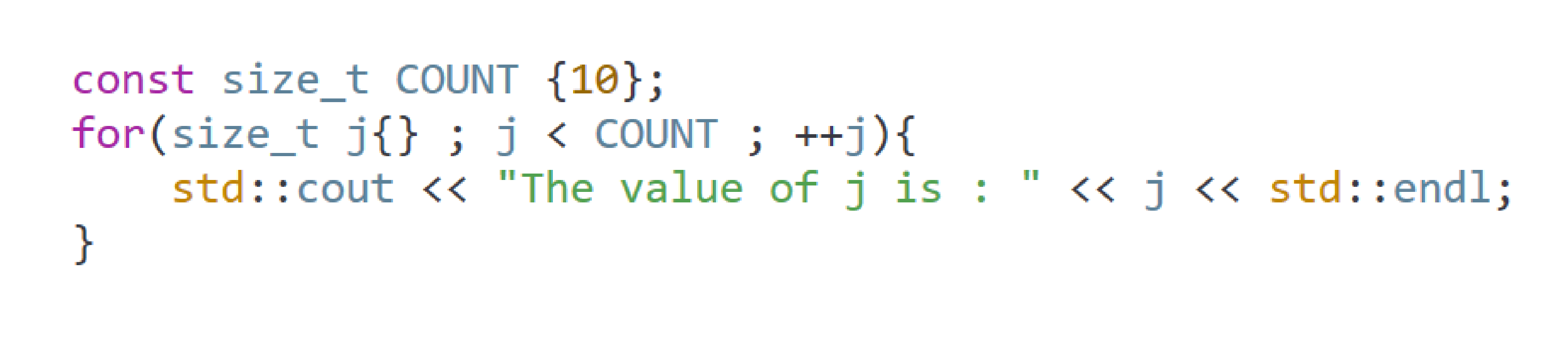
```
size_t j{} ;

for(j ; j < 10 ; ++j){
    std::cout << "Using the j variable from main function local scope : " << j << std::endl;
}
std::cout << "Loop done, the value of j is : " << j << std::endl;
```

Can leave out the iterator declaration in the loop



```
size_t j{} ;

for(  ; j < 10 ; ++j){
    std::cout << "Using the j variable from main function local scope : " << j << std::endl;
}
std::cout << "Loop done, the value of j is : " << j << std::endl;
```

Hard coded values are bad

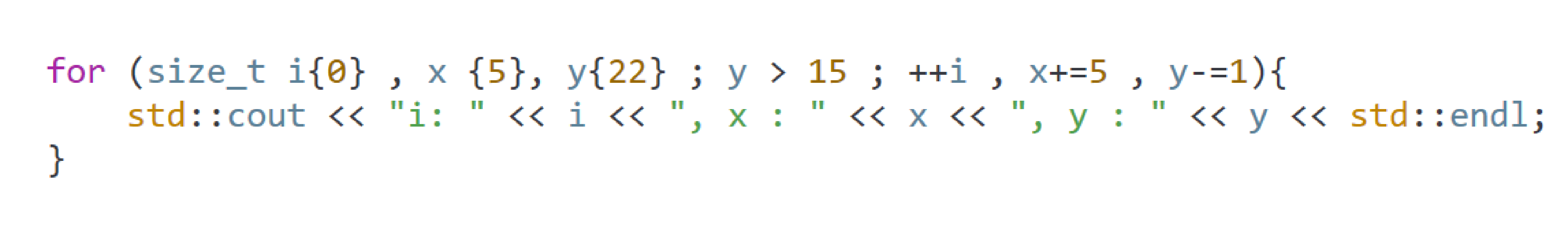


```cpp
const size_t COUNT {10};
for(size_t j{} ; j < COUNT ; ++j){
    std::cout << "The value of j is : " << j << std::endl;
}
```

Slide intentionally left empty

for loop : Multiple Declarations

Multiple Declarations

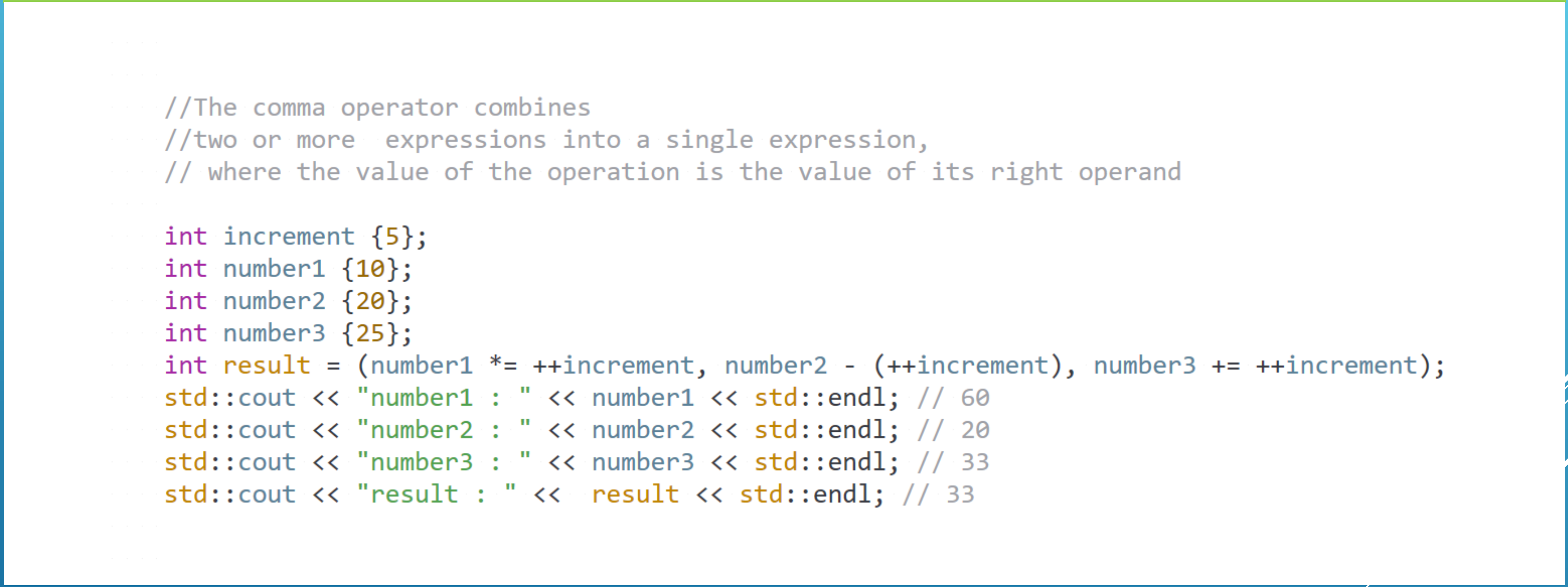


```
for (size_t i{0} , x {5}, y{22} ; y > 15 ; ++i , x+=5 , y-=1){
    std::cout << "i: " << i << ", x : " << x << ", y : " << y << std::endl;
}
```

Slide intentionally left empty

Comma Operator

Comma Operator

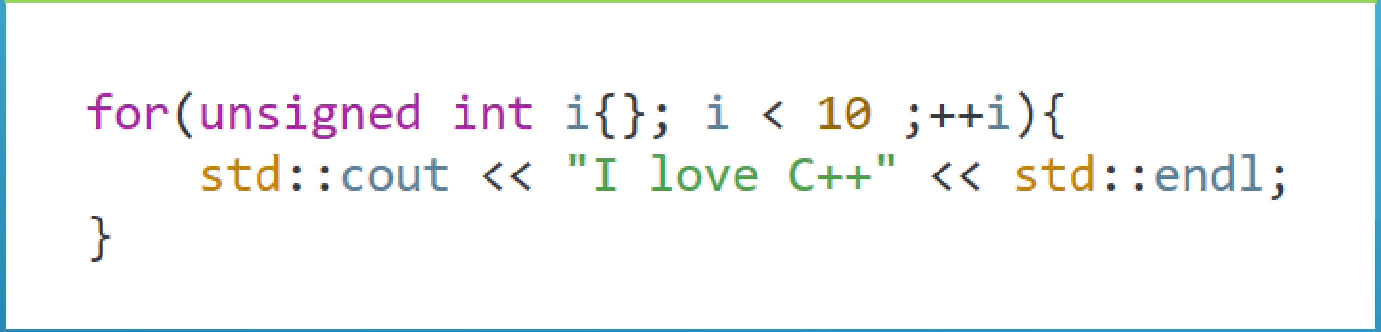


```
//The comma operator combines
//two or more  expressions into a single expression,
// where the value of the operation is the value of its right operand

int increment {5};
int number1 {10};
int number2 {20};
int number3 {25};
int result = (number1 *= ++increment, number2 - (++increment), number3 += ++increment);
std::cout << "number1 : " << number1 << std::endl; // 60
std::cout << "number2 : " << number2 << std::endl; // 20
std::cout << "number3 : " << number3 << std::endl; // 33
std::cout << "result : " <<  result << std::endl; // 33
```

Slide intentionally left empty

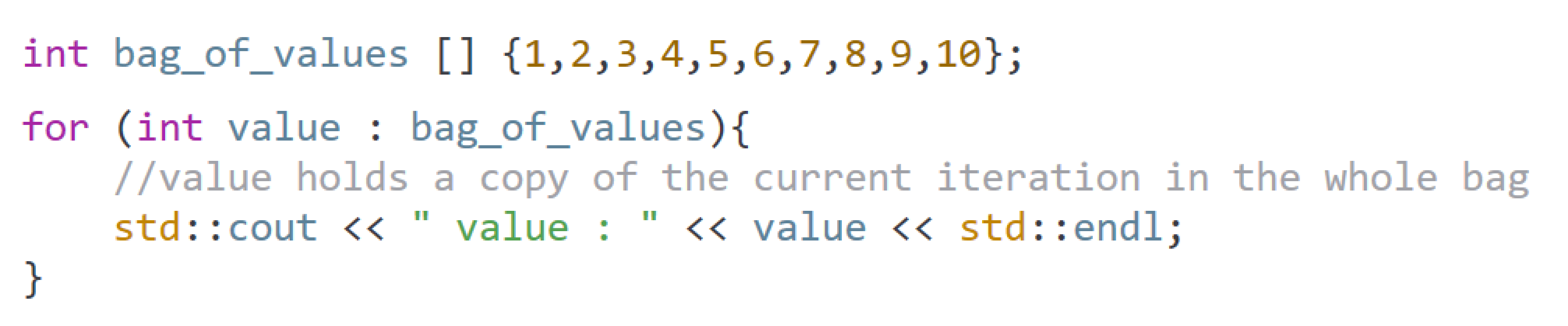
Range based for loop

for loop



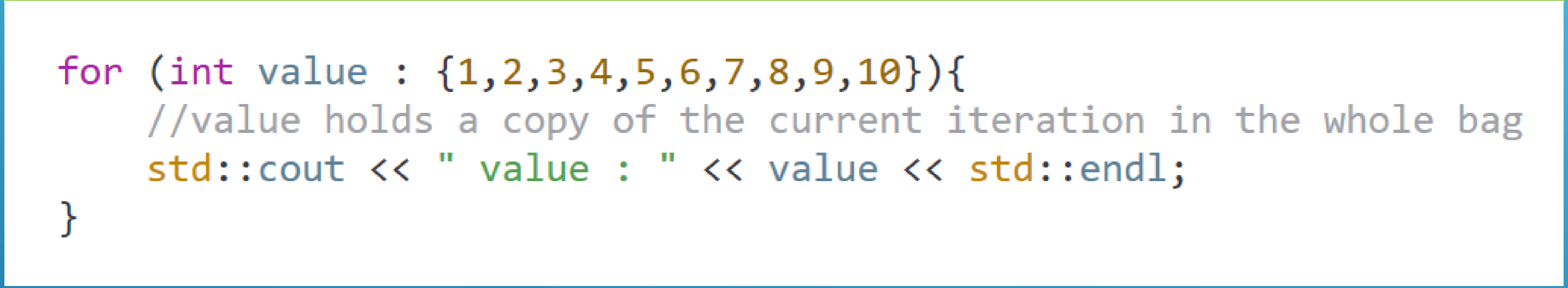
```cpp
for(unsigned int i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

Range based for loop



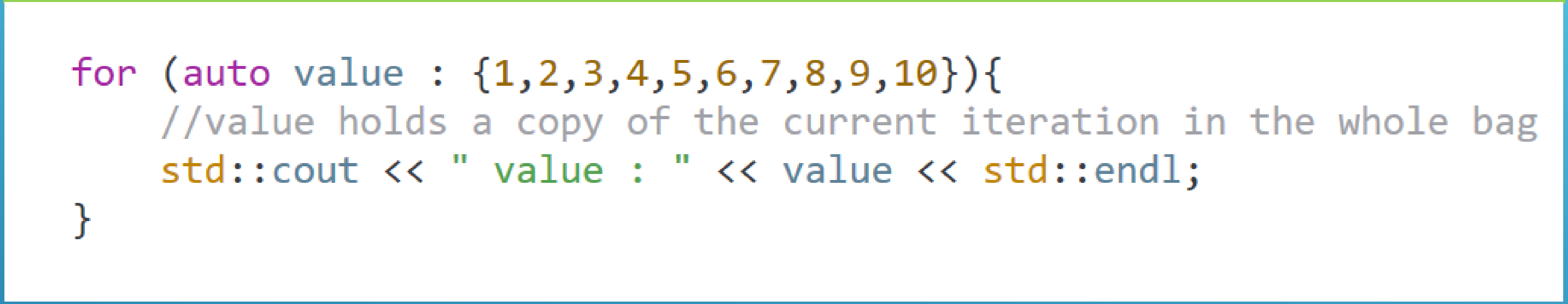
```cpp
int bag_of_values [] {1,2,3,4,5,6,7,8,9,10};

for (int value : bag_of_values){
    //value holds a copy of the current iteration in the whole bag
    std::cout << " value : " << value << std::endl;
}
```

Specify the collection in place



```cpp
for (int value : {1,2,3,4,5,6,7,8,9,10}){
    //value holds a copy of the current iteration in the whole bag
    std::cout << " value : " << value << std::endl;
}
```

Let the compiler deduce the type



```cpp
for (auto value : {1,2,3,4,5,6,7,8,9,10}){
    //value holds a copy of the current iteration in the whole bag
    std::cout << " value : " << value << std::endl;
}
```

scope	Value is scoped within the for block. You can’t use it from the outside of the for block.

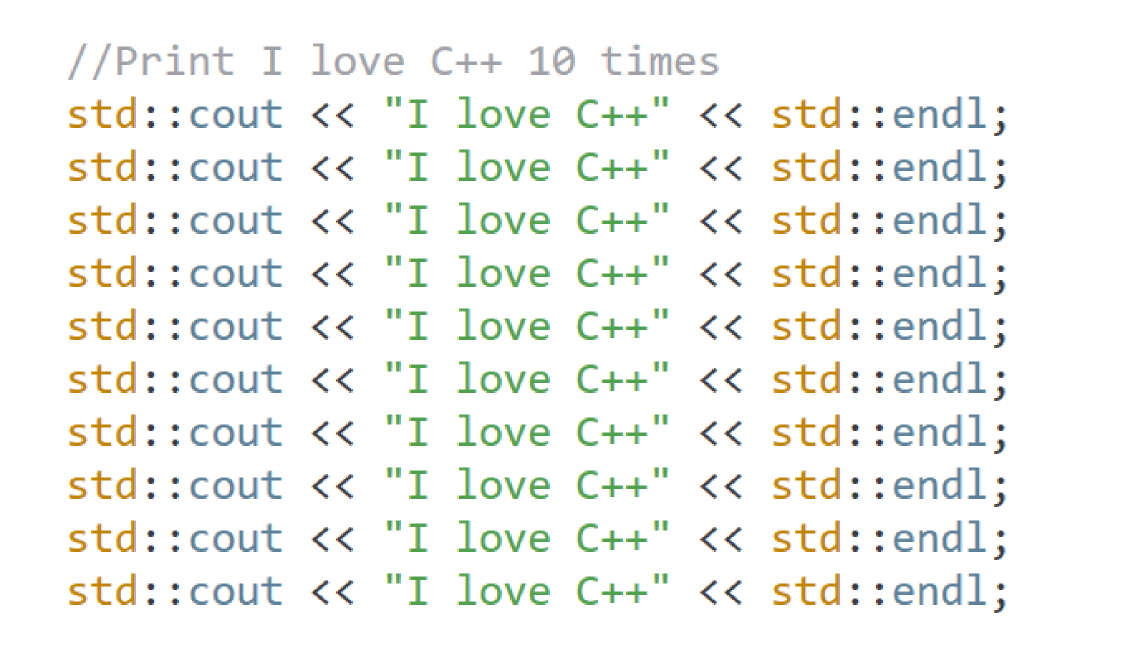
Pillars of any loop

- Iterator
- Starting Point
- Test(controls when the loop stops)
- Increment(Decrement)
- Loop body

Slide intentionally left empty

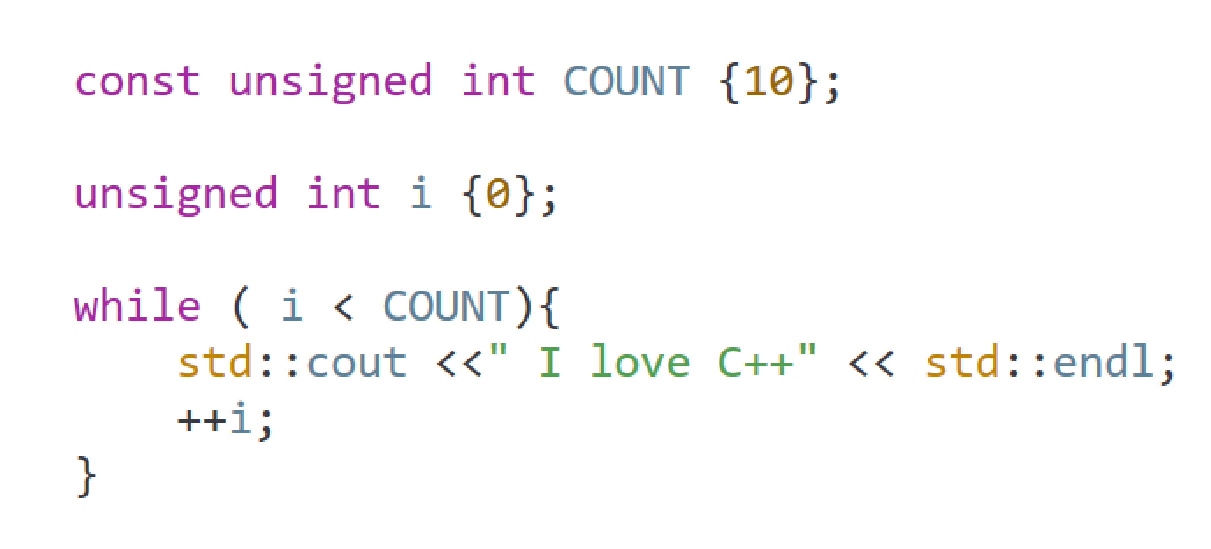
While loop

Repetitive tasks



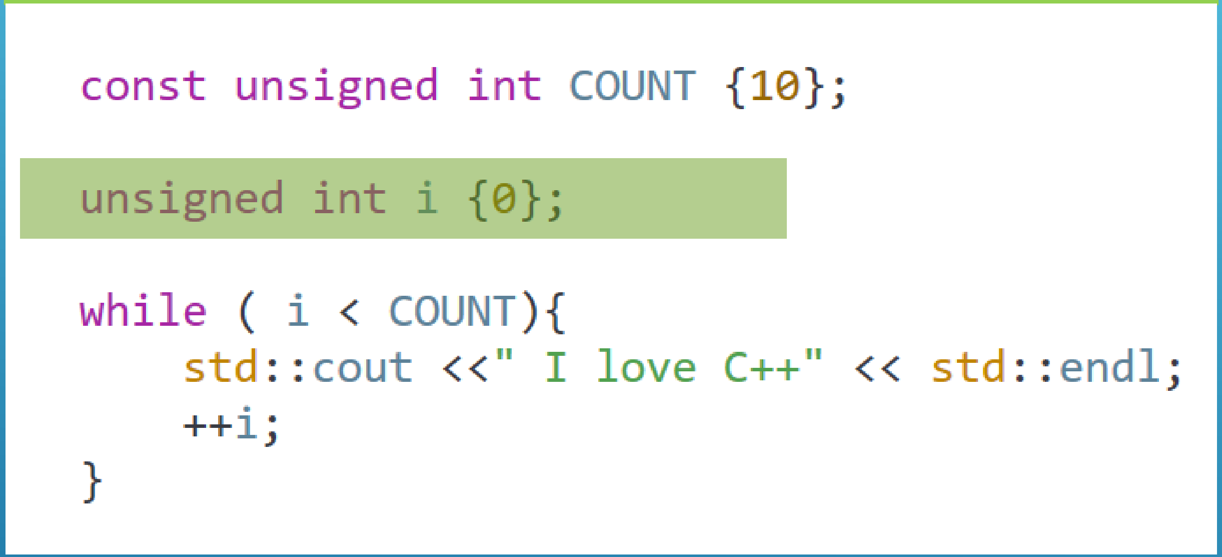
```
//Print I love C++ 10 times
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
```

while loop



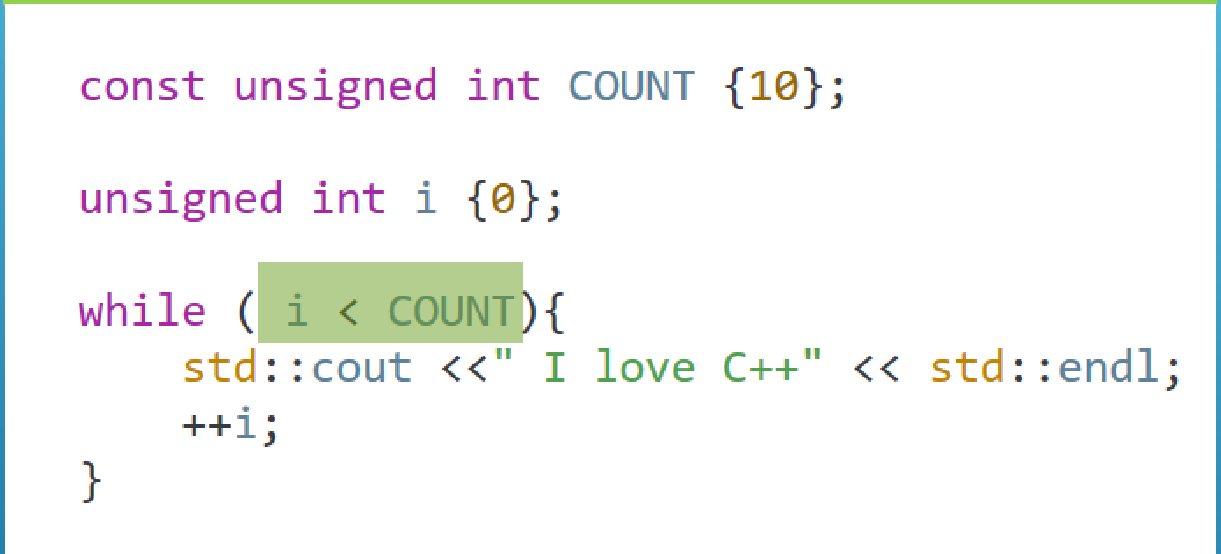
```
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

while loop



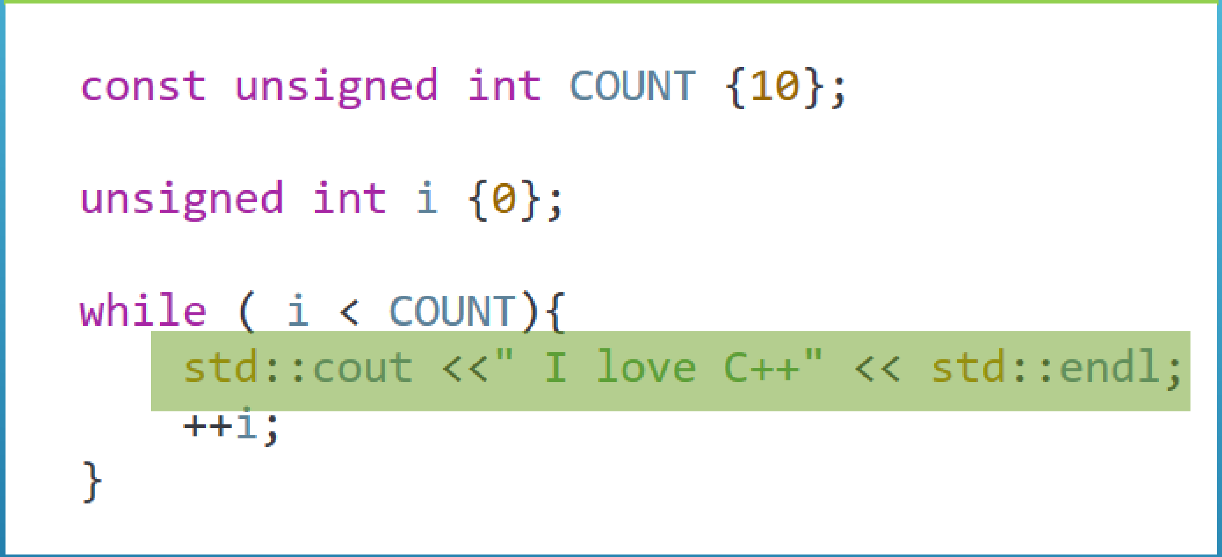
```
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

while loop

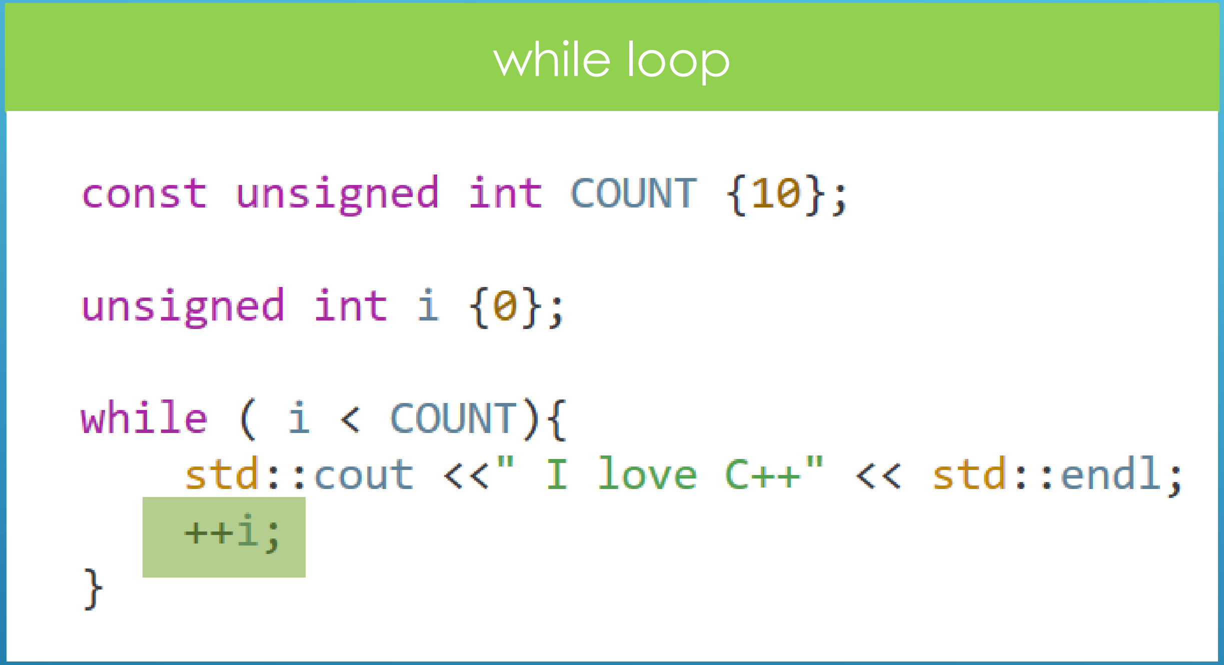


```
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

while loop



```
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```



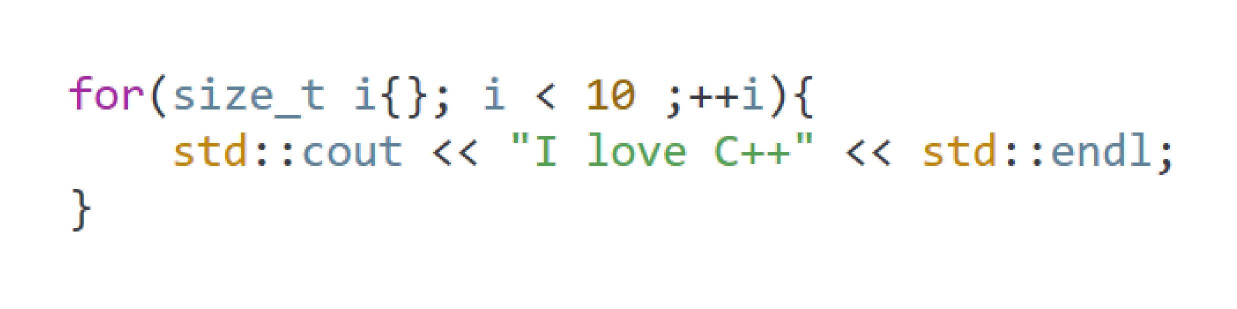
while loop

```
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

Pillars of any loop

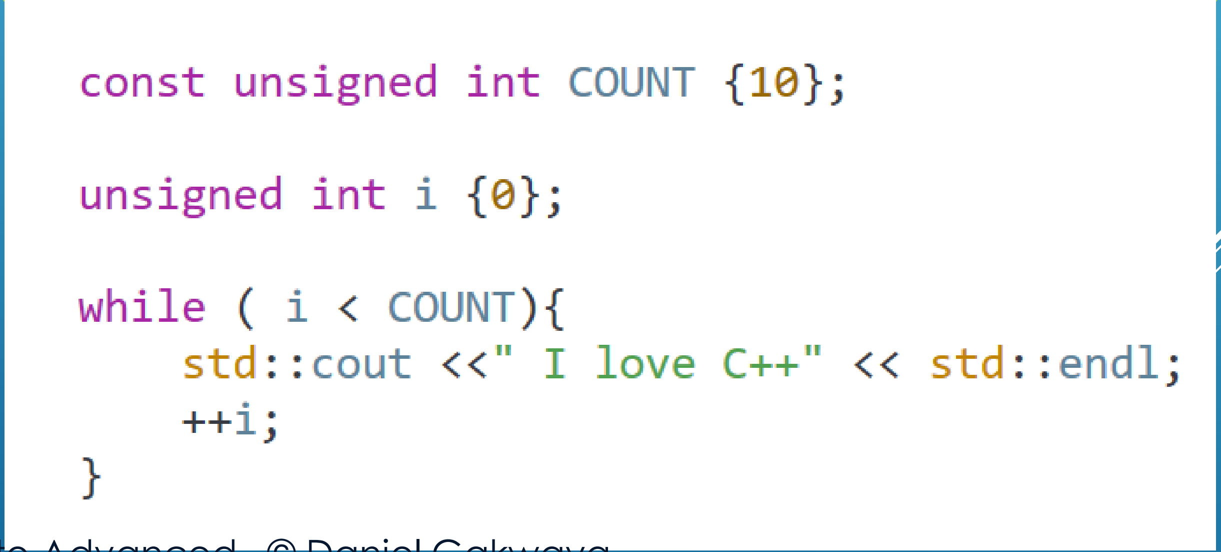
- Iterator
- Starting Point
- Test(controls when the loop stops)
- Increment(Decrement)
- Loop body

for loop



```cpp
for(size_t i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

while loop

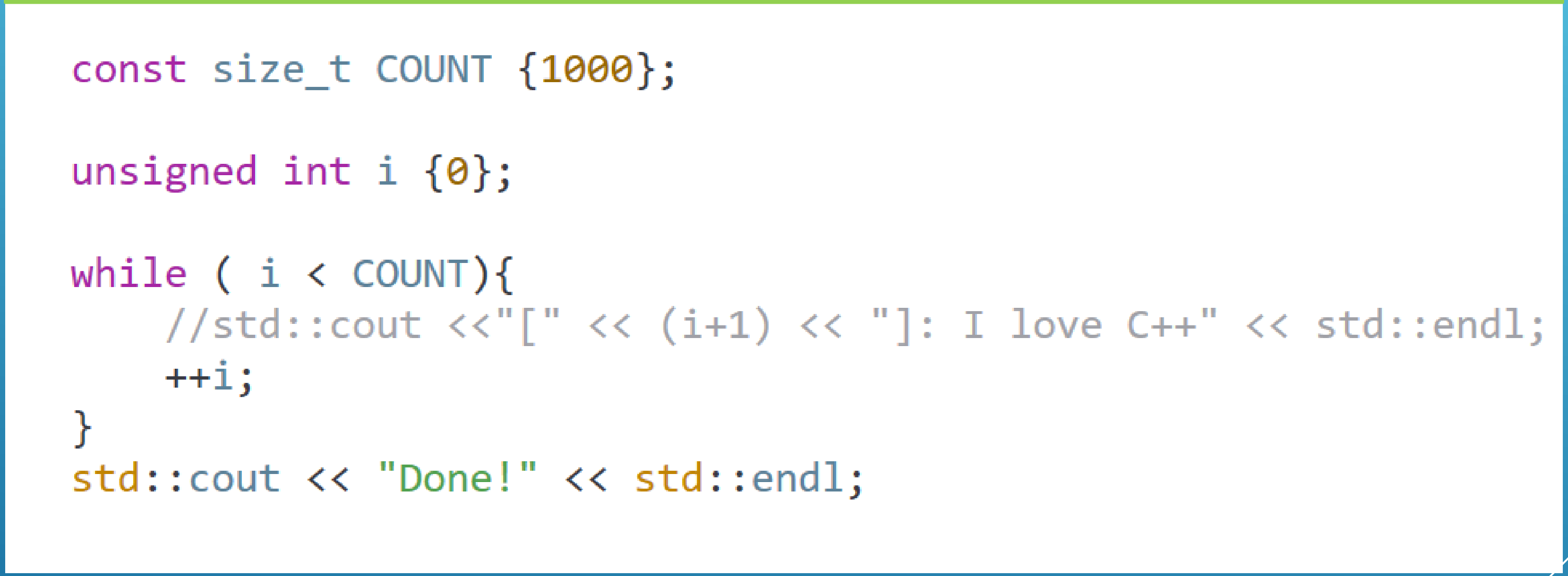


```cpp
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

Slide intentionally left empty

Huge Loops with Output

Loop without output



```
const size_t COUNT {1000};

unsigned int i {0};

while ( i < COUNT){
    //std::cout <<"[" << (i+1) << "]: I love C++" << std::endl;
    ++i;
}
std::cout << "Done!" << std::endl;
```

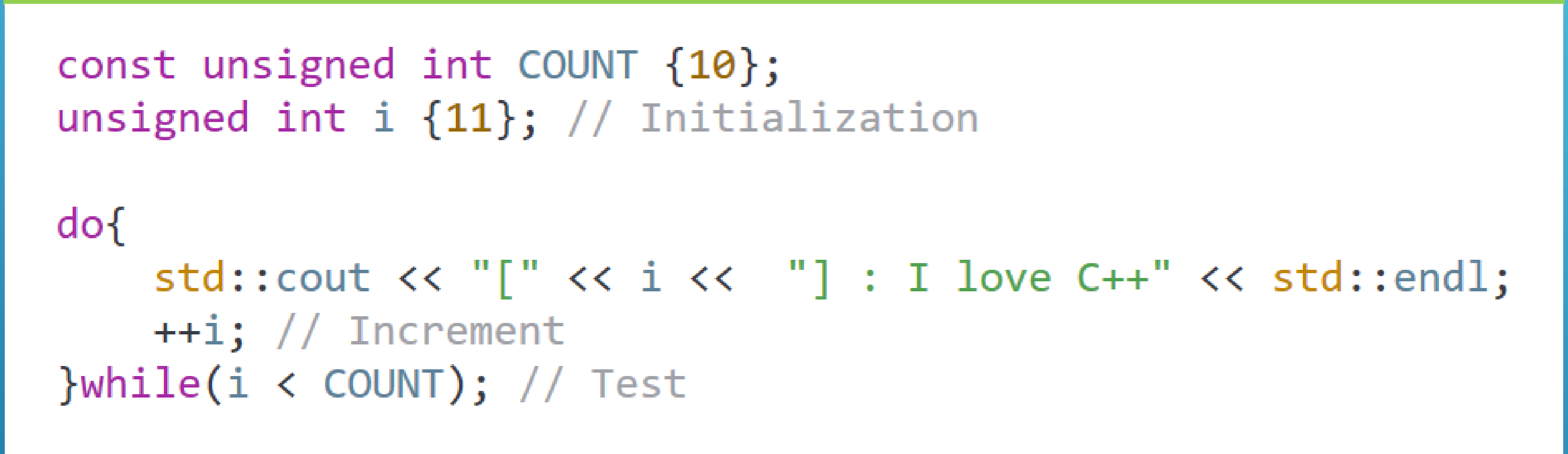
Loop with output

```
const size_t COUNT {1000};

for(size_t i{}; i < COUNT ; ++i){
    std::cout <<"[" << (i+1) << "]: I love C++" << std::endl;
}
std::cout << "Done!" << std::endl;
```

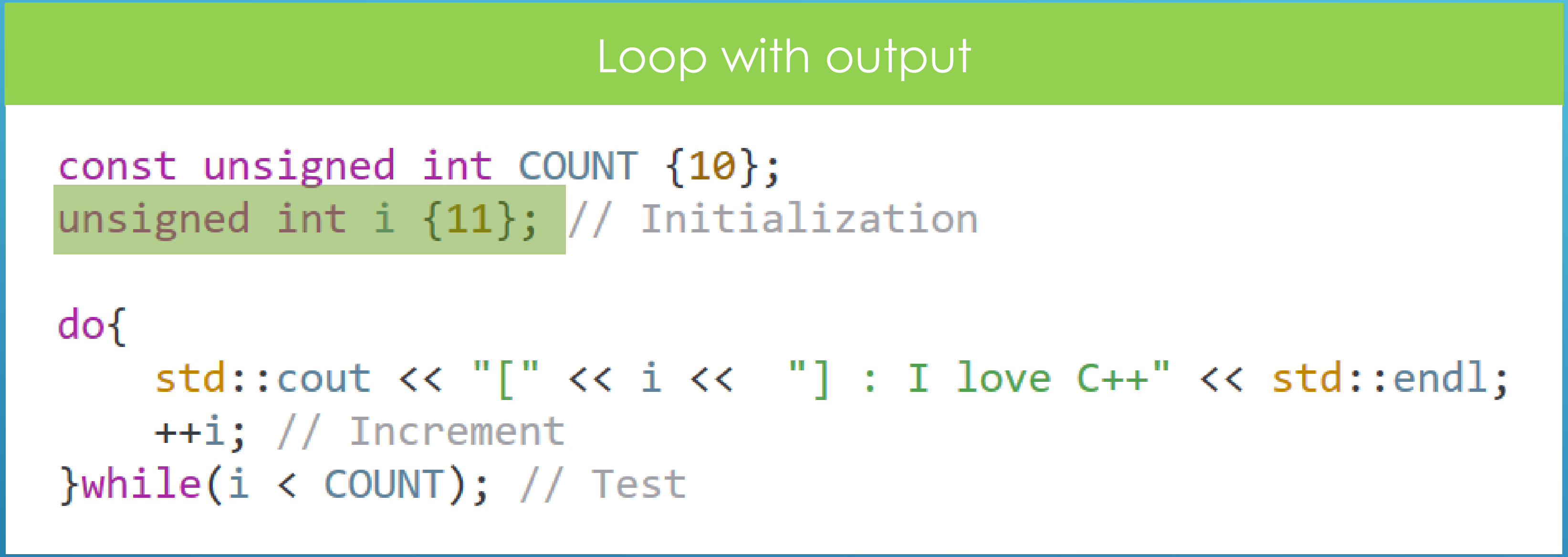
Slide intentionally left empty

Do while loops

Loop with output



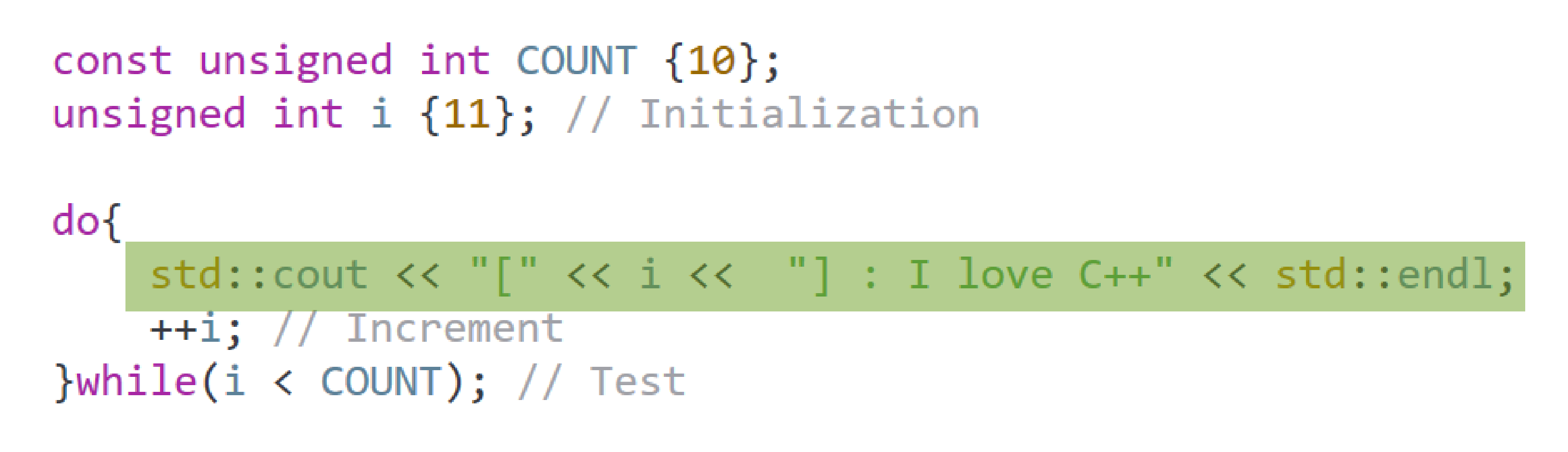
```
const unsigned int COUNT {10};
unsigned int i {11}; // Initialization

do{
    std::cout << "[" << i <<  "] : I love C++" << std::endl;
    ++i; // Increment
}while(i < COUNT); // Test
```



Loop with output

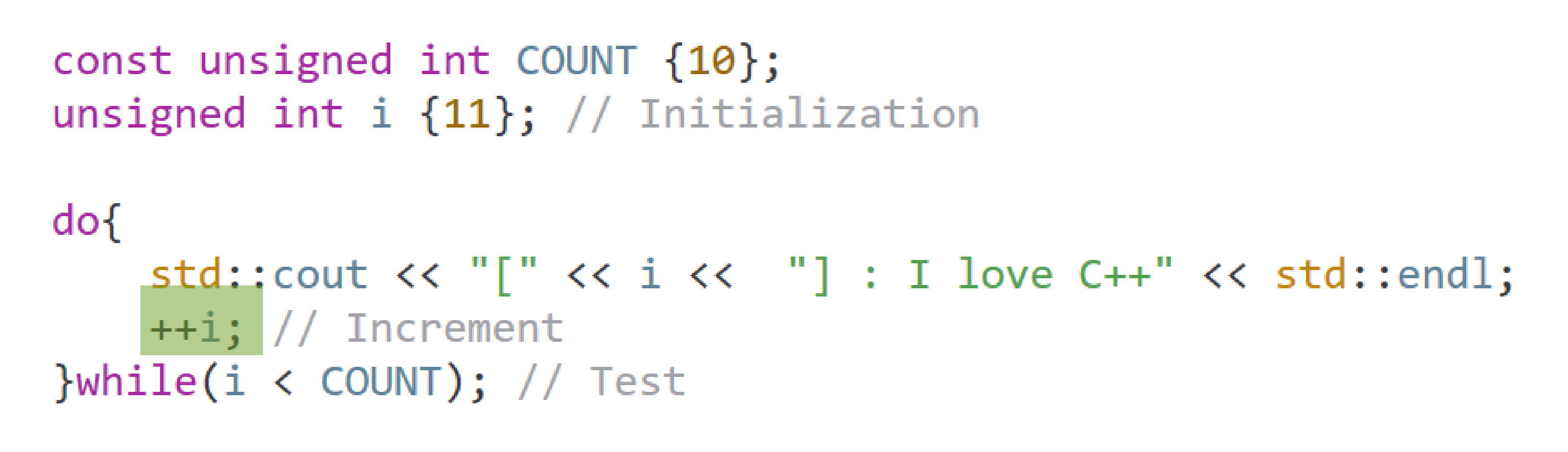
```
const unsigned int COUNT {10};
unsigned int i {11}; // Initialization

do{
    std::cout << "[" << i <<  "] : I love C++" << std::endl;
    ++i; // Increment
}while(i < COUNT); // Test
```

Loop with output



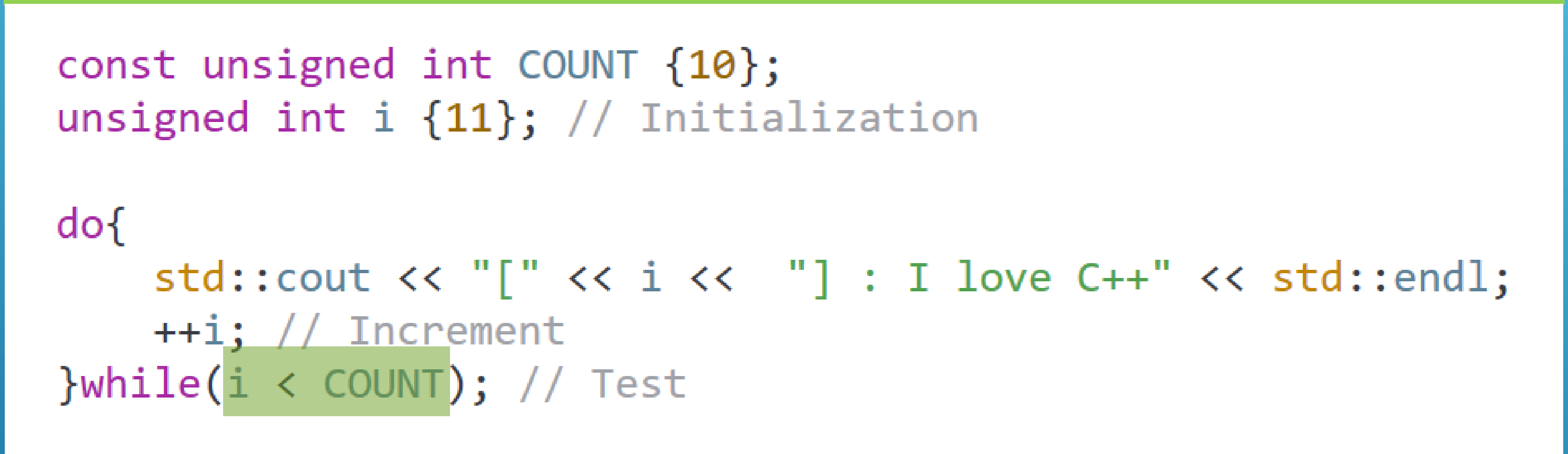
```
const unsigned int COUNT {10};
unsigned int i {11}; // Initialization

do{
    std::cout << "[" << i <<  "] : I love C++" << std::endl;
    ++i; // Increment
}while(i < COUNT); // Test
```

Loop with output



```
const unsigned int COUNT {10};
unsigned int i {11}; // Initialization

do{
    std::cout << "[" << i <<  "] : I love C++" << std::endl;
    ++i; // Increment
}while(i < COUNT); // Test
```

Loop with output



```
const unsigned int COUNT {10};
unsigned int i {11}; // Initialization

do{
    std::cout << "[" << i <<  "] : I love C++" << std::endl;
    ++i; // Increment
}while(i < COUNT); // Test
```

Pillars of any loop

- Iterator
- Starting Point
- Test(controls when the loop stops)
- Increment(Decrement)
- Loop body

do while loop

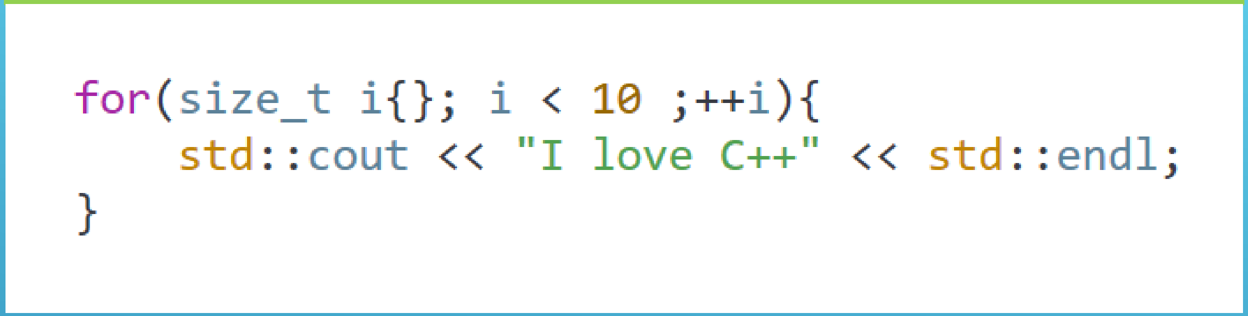
Runs the body then checks

Slide intentionally left empty

Infinite Loops

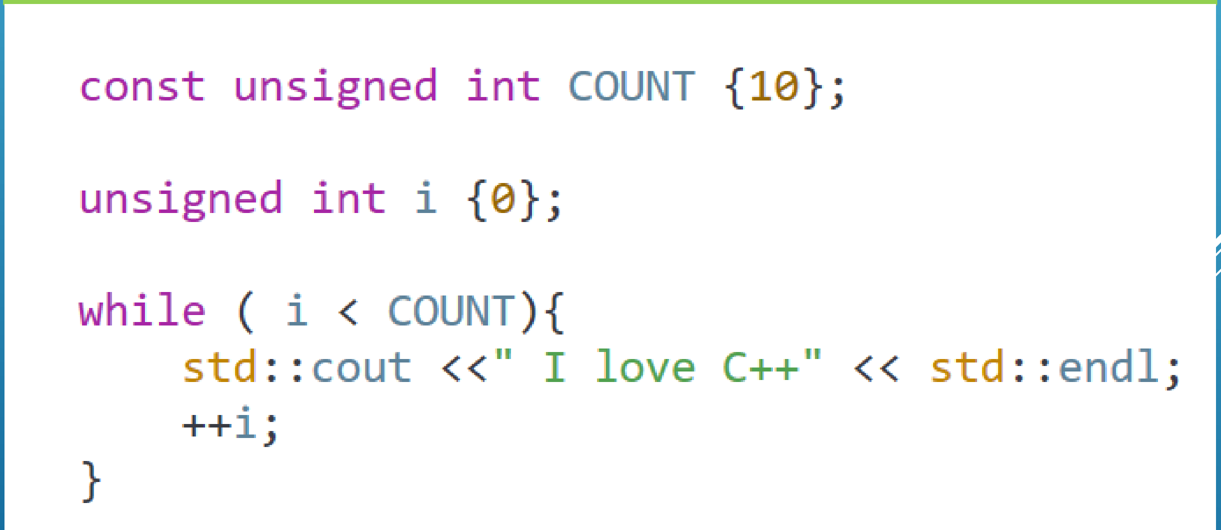
Loops that run indefinitely and never stop

for loop



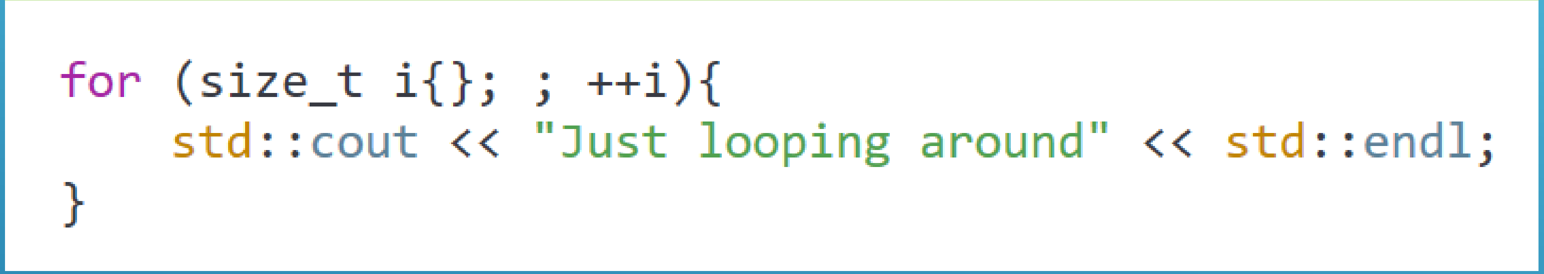
```cpp
for(size_t i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

while loop



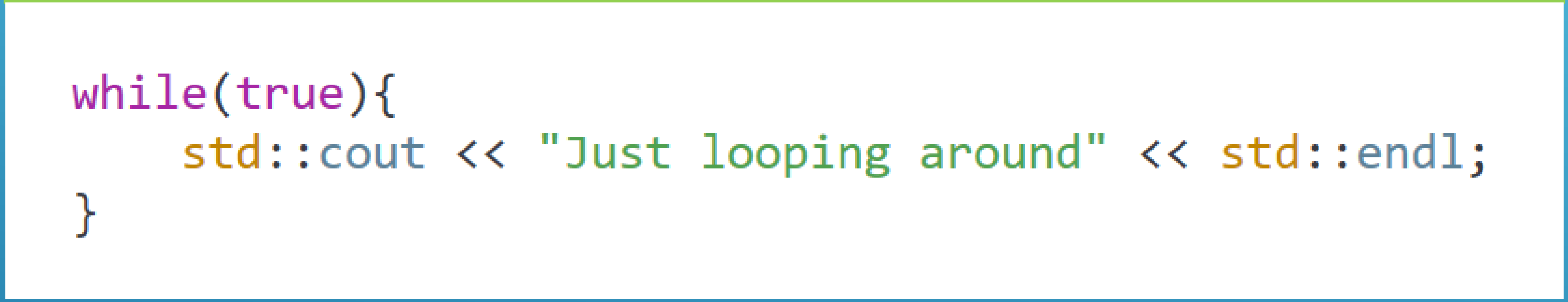
```cpp
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

Infinite for loop



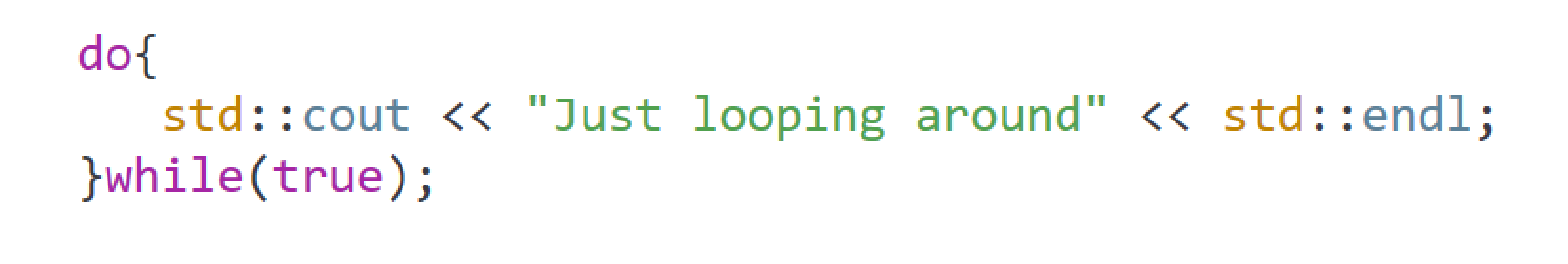
```cpp
for (size_t i{}; ; ++i){
    std::cout << "Just looping around" << std::endl;
}
```

Infinite while loop



```
while(true){
    std::cout << "Just looping around" << std::endl;
}
```

Infinite do while loop



```
do{
   std::cout << "Just looping around" << std::endl;
}while(true);
```

Infinite loops

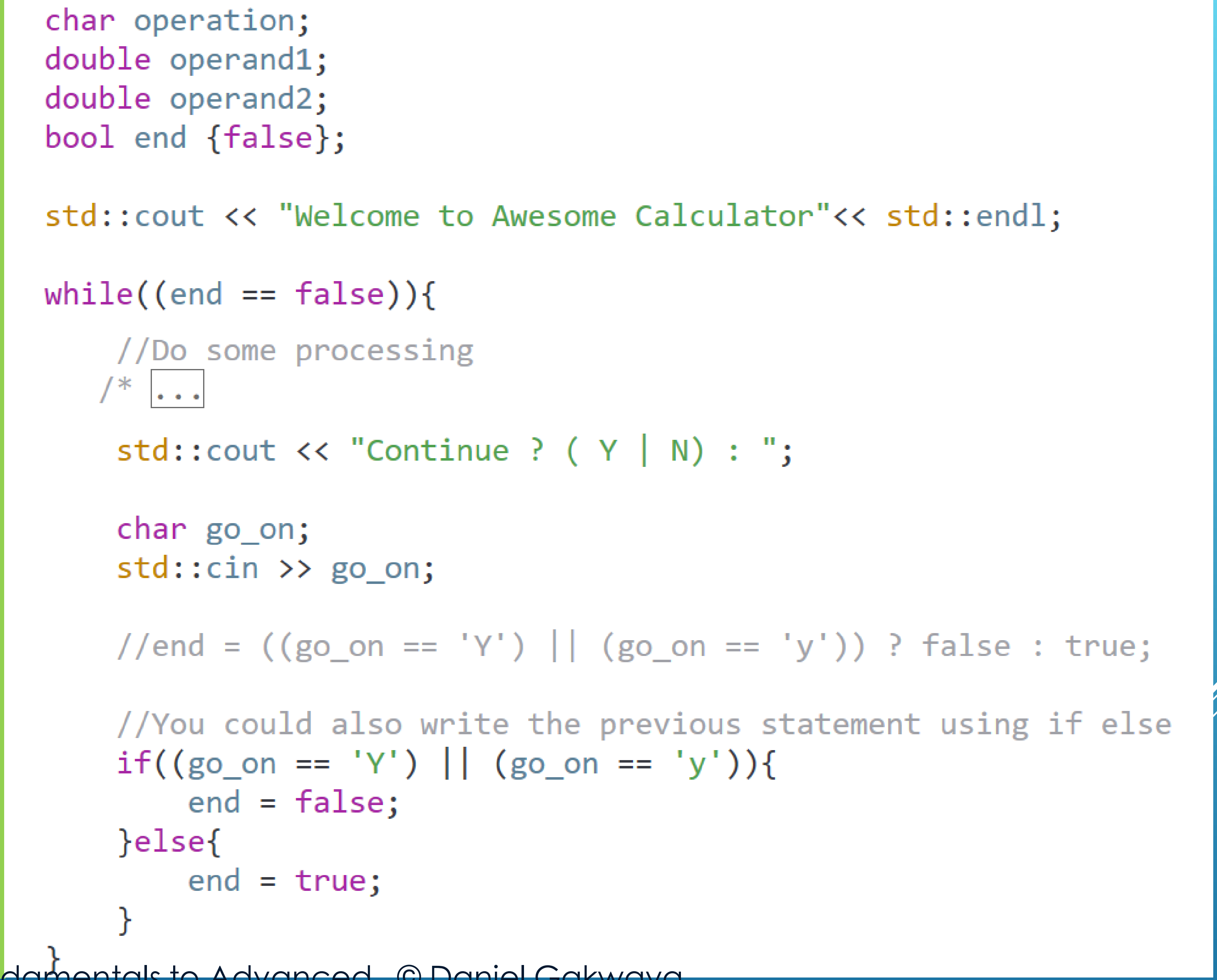
- They sometimes occur by error an may cause your program to do crazy things. Watch out for this!
- Sometimes they are just part of your design especially when you don’t now how many times your loop will execute, when that’s determined by the user for example. We’ll see an example about that in the next lecture.

Slide intentionally left empty

Infinite Loops : Practice

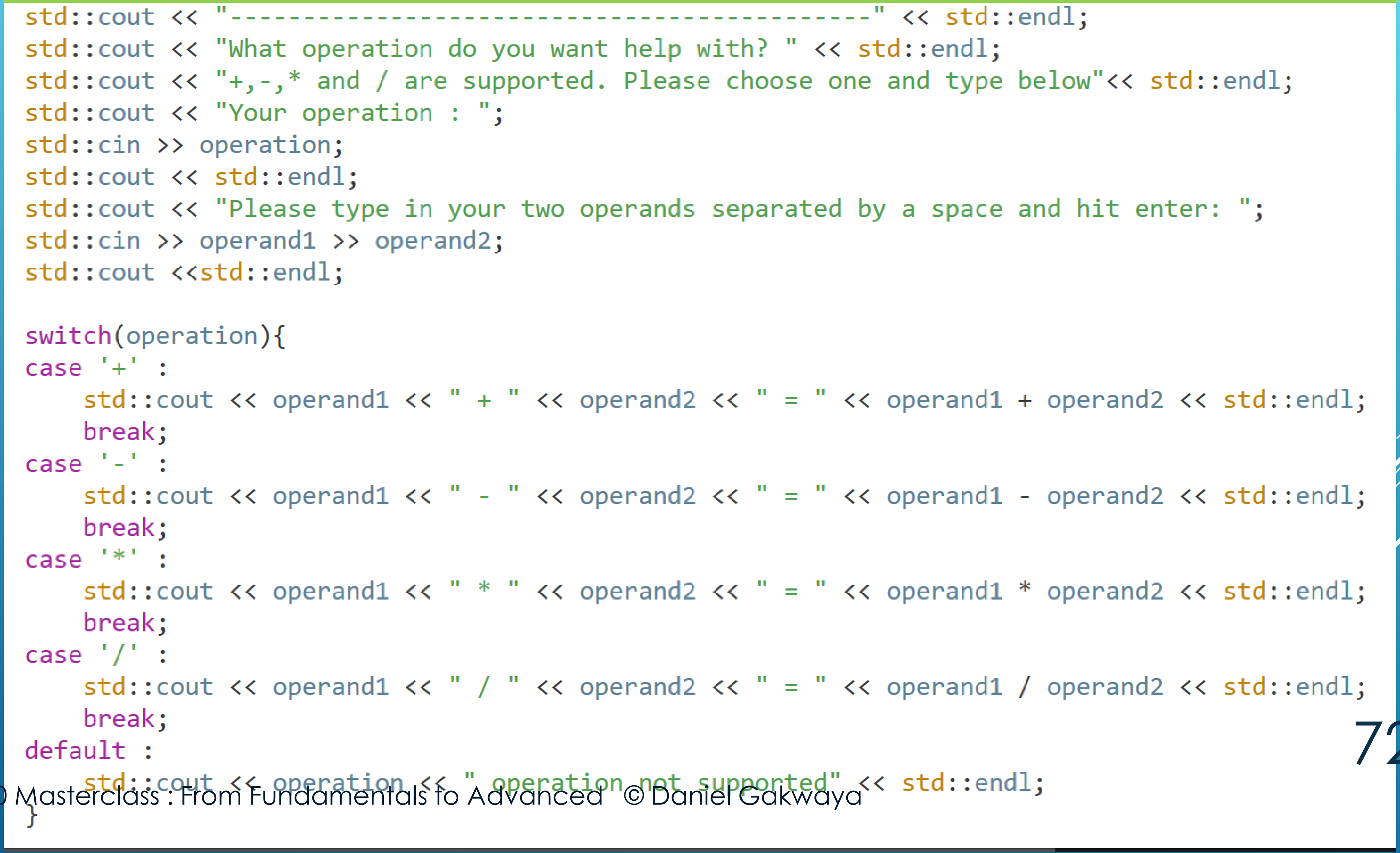
A terminal based calculator

User decides when the loop ends



```
char operation;
double operand1;
double operand2;
bool end {false};

std::cout << "Welcome to Awesome Calculator"<< std::endl;

while((end == false)){
    //Do some processing
   /* ...
    std::cout << "Continue ? ( Y | N) : ";

    char go_on;
    std::cin >> go_on;

    //end = ((go_on == 'Y') || (go_on == 'y')) ? false : true;

    //You could also write the previous statement using if else
    if((go_on == 'Y') || (go_on == 'y')){
        end = false;
    }else{
        end = true;
    }
}
```

Processing

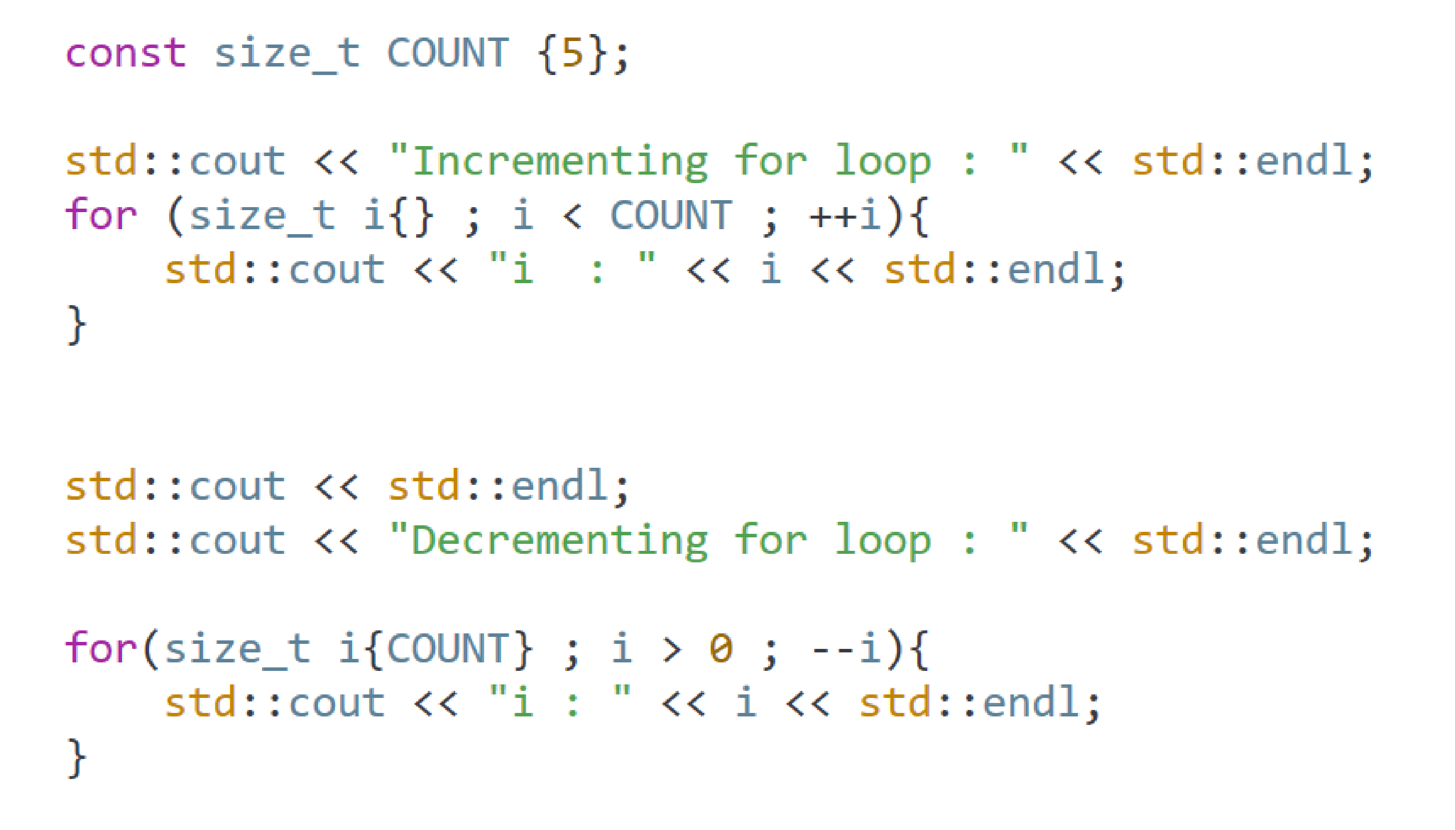


```
std::cout << "-----------------------------------------------" << std::endl;
std::cout << "What operation do you want help with? " << std::endl;
std::cout << "+,-,* and / are supported. Please choose one and type below"<< std::endl;
std::cout << "Your operation : ";
std::cin >> operation;
std::cout << std::endl;
std::cout << "Please type in your two operands separated by a space and hit enter: ";
std::cin >> operand1 >> operand2;
std::cout <<std::endl;

switch(operation){
case '+' :
    std::cout << operand1 << " + " << operand2 << " = " << operand1 + operand2 << std::endl;
    break;
case '-' :
    std::cout << operand1 << " - " << operand2 << " = " << operand1 - operand2 << std::endl;
    break;
case '*' :
    std::cout << operand1 << " * " << operand2 << " = " << operand1 * operand2 << std::endl;
    break;
case '/' :
    std::cout << operand1 << " / " << operand2 << " = " << operand1 / operand2 << std::endl;
    break;
default :
    std::cout << operation << " operation not supported" << std::endl;
}
```

Slide intentionally left empty

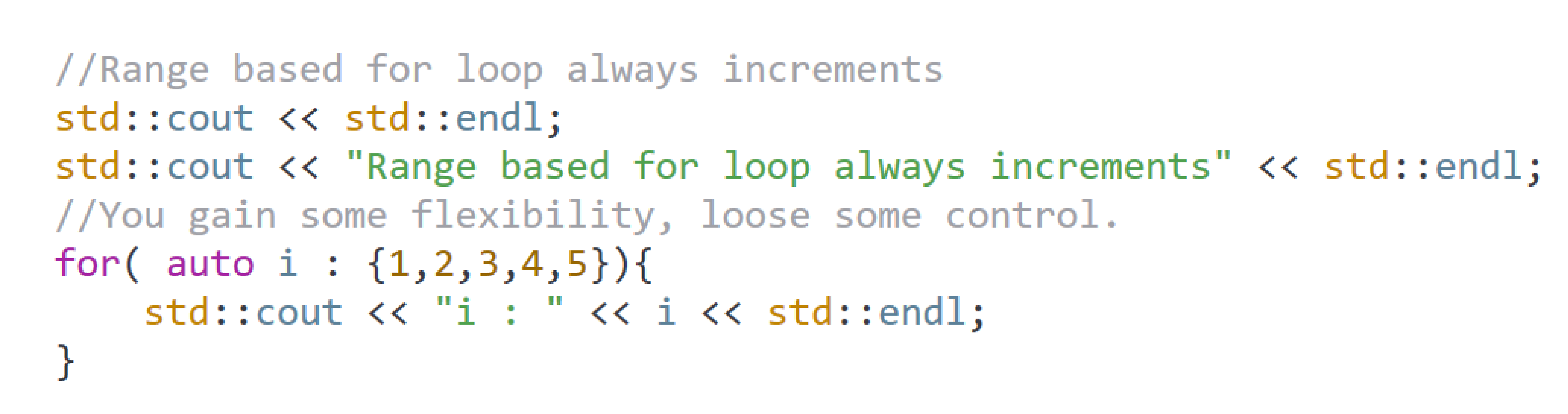
Decrementing Loops

for loop



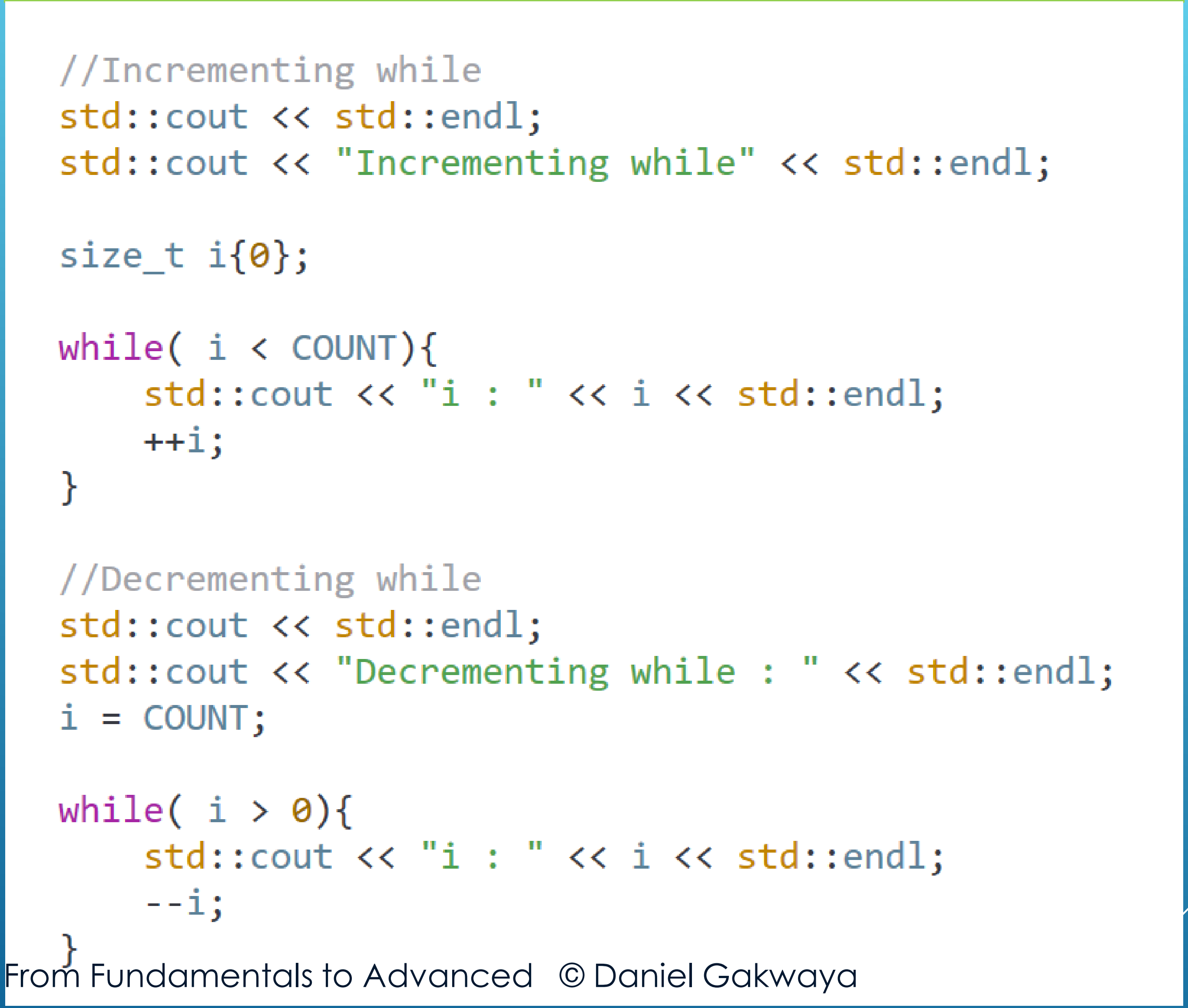
```cpp
const size_t COUNT {5};

std::cout << "Incrementing for loop : " << std::endl;
for (size_t i{} ; i < COUNT ; ++i){
    std::cout << "i  : " << i << std::endl;
}


std::cout << std::endl;
std::cout << "Decrementing for loop : " << std::endl;

for(size_t i{COUNT} ; i > 0 ; --i){
    std::cout << "i : " << i << std::endl;
}
```

Range based for loop



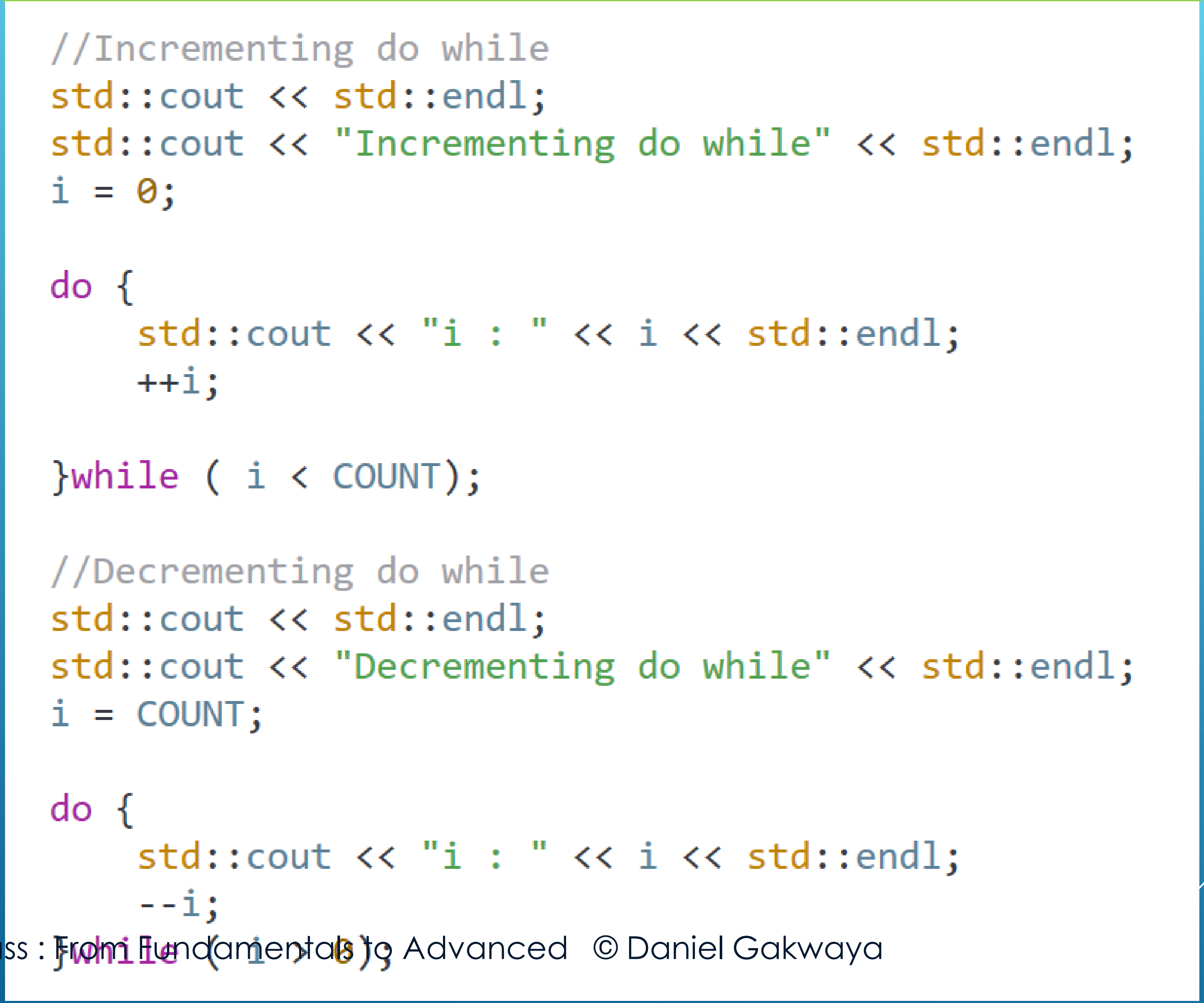
```
//Range based for loop always increments
std::cout << std::endl;
std::cout << "Range based for loop always increments" << std::endl;
//You gain some flexibility, loose some control.
for( auto i : {1,2,3,4,5}){
    std::cout << "i : " << i << std::endl;
}
```

While loop



```
//Incrementing while
std::cout << std::endl;
std::cout << "Incrementing while" << std::endl;

size_t i{0};

while( i < COUNT){
    std::cout << "i : " << i << std::endl;
    ++i;
}

//Decrementing while
std::cout << std::endl;
std::cout << "Decrementing while : " << std::endl;
i = COUNT;

while( i > 0){
    std::cout << "i : " << i << std::endl;
    --i;
}
```

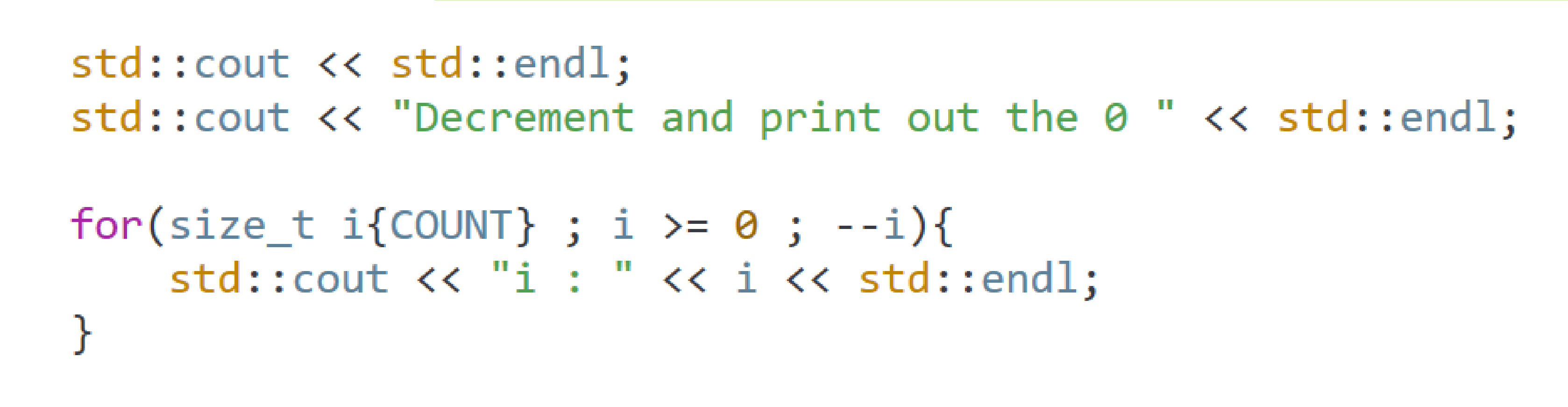
do while loop



```
//Incrementing do while
std::cout << std::endl;
std::cout << "Incrementing do while" << std::endl;
i = 0;

do {
    std::cout << "i : " << i << std::endl;
    ++i;

}while ( i < COUNT);

//Decrementing do while
std::cout << std::endl;
std::cout << "Decrementing do while" << std::endl;
i = COUNT;

do {
    std::cout << "i : " << i << std::endl;
    --i;
}while ( i > 0);
```

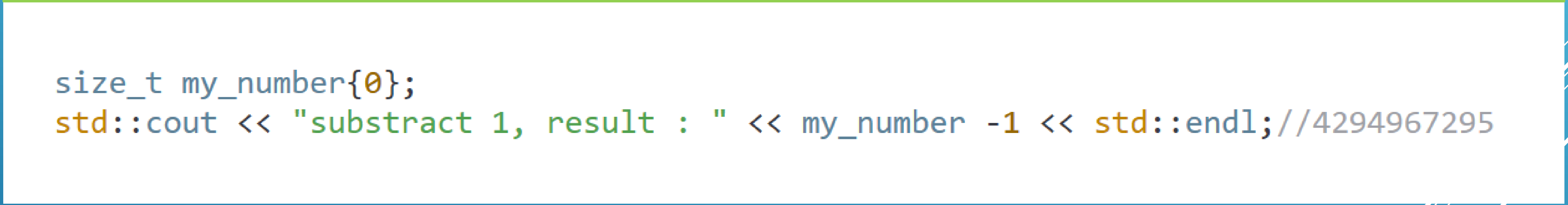
DANGER !

Underflow and infinite loop



```
std::cout << std::endl;
std::cout << "Decrement and print out the 0 " << std::endl;

for(size_t i{COUNT} ; i >= 0 ; --i){
    std::cout << "i : " << i << std::endl;
}
```

Underflow in action



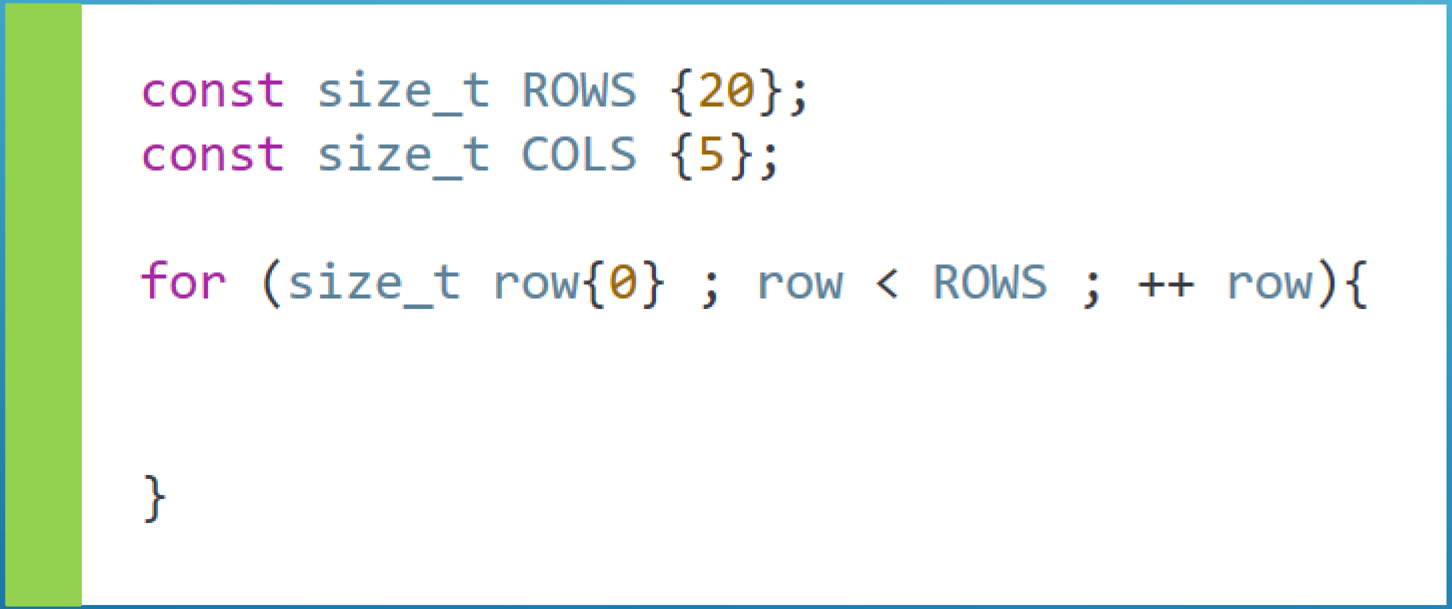
```
size_t my_number{0};
std::cout << "substract 1, result : " << my_number -1 << std::endl;//4294967295
```

Slide intentionally left empty

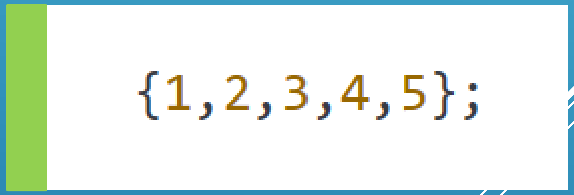
Nested Loops

Nested Loops

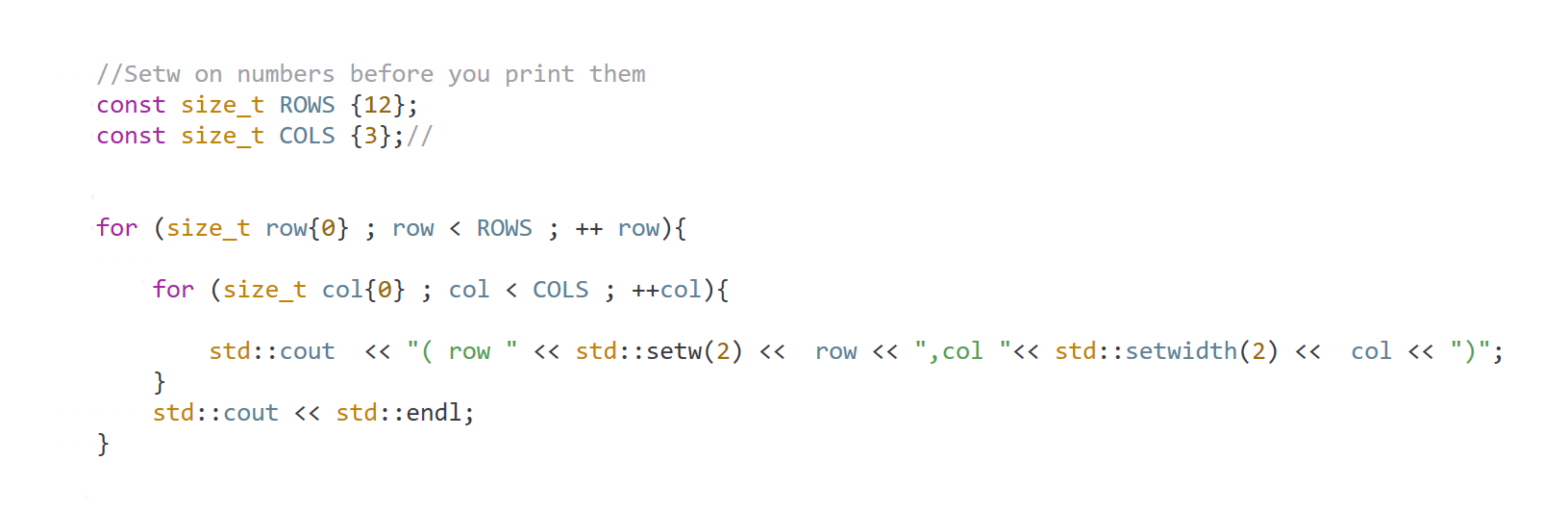
The more loops you nest, the more dimensions the data you're looping around could represent.



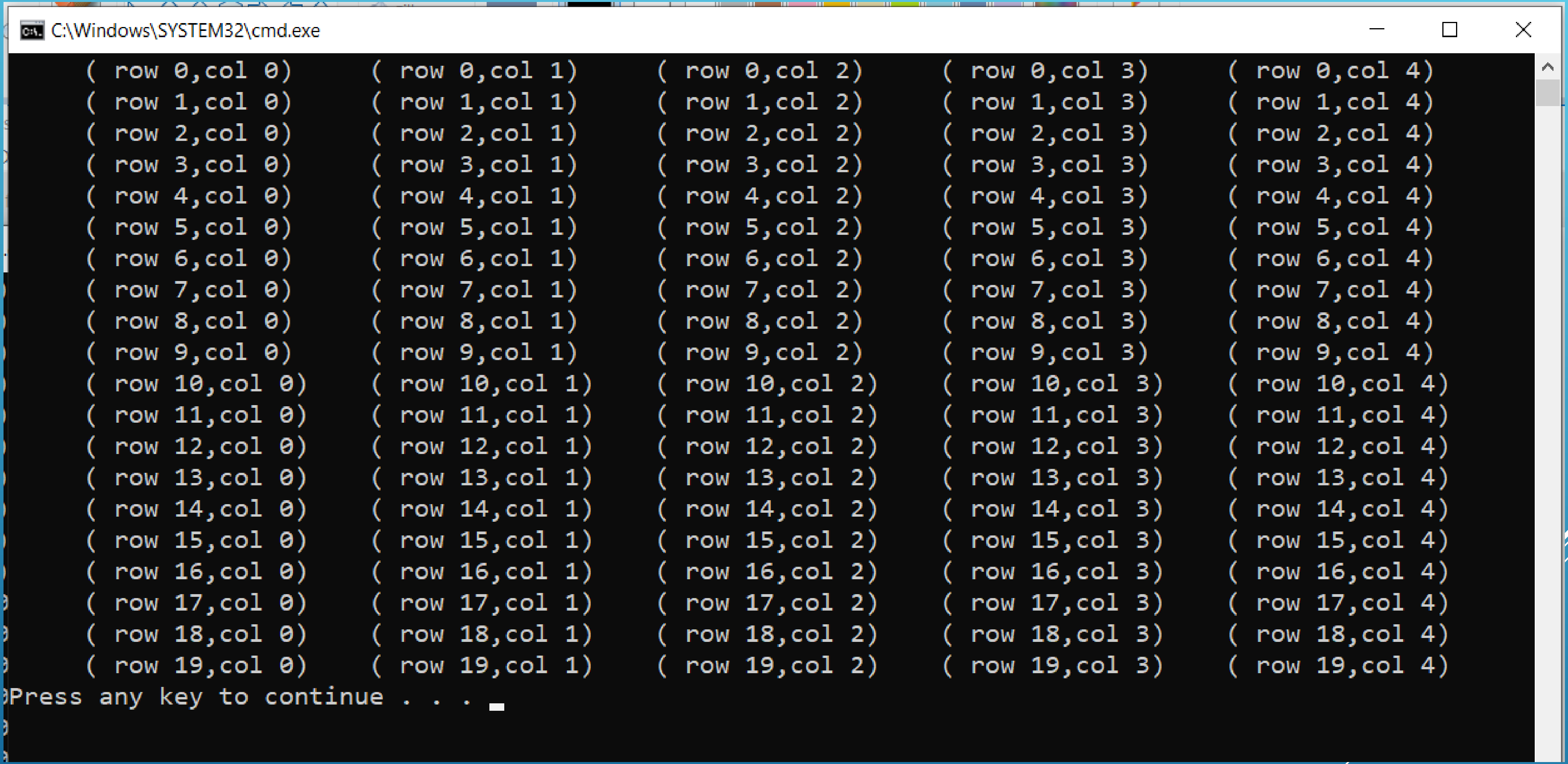
```cpp
const size_t ROWS {20};
const size_t COLS {5};

for (size_t row{0} ; row < ROWS ; ++ row){


}
```



```cpp
{1,2,3,4,5};
```

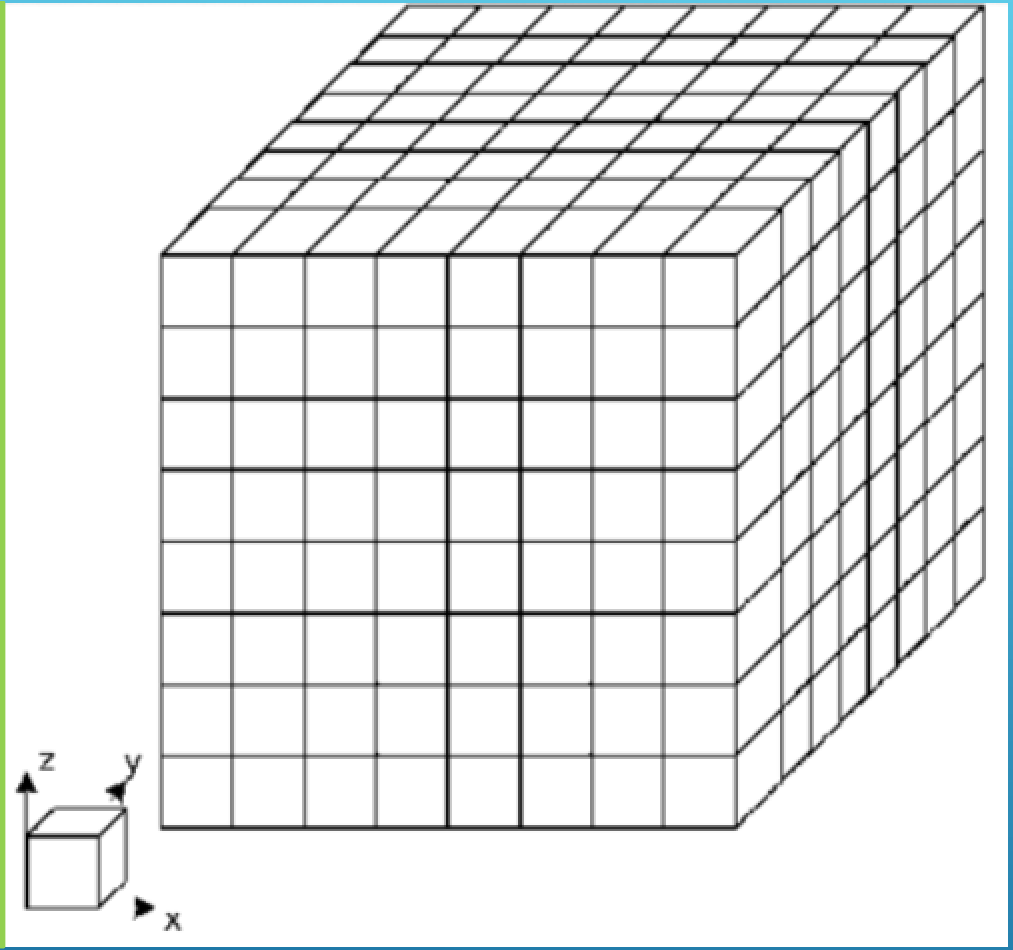


```
//Setw on numbers before you print them
const size_t ROWS {12};
const size_t COLS {3};//


for (size_t row{0} ; row < ROWS ; ++ row){

    for (size_t col{0} ; col < COLS ; ++col){

        std::cout  << "( row " << std::setw(2) <<  row << ",col "<< std::setwidth(2) <<  col << ")";
    }
    std::cout << std::endl;
}
```



```
     ( row 0,col 0)     ( row 0,col 1)     ( row 0,col 2)     ( row 0,col 3)     ( row 0,col 4)
     ( row 1,col 0)     ( row 1,col 1)     ( row 1,col 2)     ( row 1,col 3)     ( row 1,col 4)
     ( row 2,col 0)     ( row 2,col 1)     ( row 2,col 2)     ( row 2,col 3)     ( row 2,col 4)
     ( row 3,col 0)     ( row 3,col 1)     ( row 3,col 2)     ( row 3,col 3)     ( row 3,col 4)
     ( row 4,col 0)     ( row 4,col 1)     ( row 4,col 2)     ( row 4,col 3)     ( row 4,col 4)
     ( row 5,col 0)     ( row 5,col 1)     ( row 5,col 2)     ( row 5,col 3)     ( row 5,col 4)
     ( row 6,col 0)     ( row 6,col 1)     ( row 6,col 2)     ( row 6,col 3)     ( row 6,col 4)
     ( row 7,col 0)     ( row 7,col 1)     ( row 7,col 2)     ( row 7,col 3)     ( row 7,col 4)
     ( row 8,col 0)     ( row 8,col 1)     ( row 8,col 2)     ( row 8,col 3)     ( row 8,col 4)
     ( row 9,col 0)     ( row 9,col 1)     ( row 9,col 2)     ( row 9,col 3)     ( row 9,col 4)
     ( row 10,col 0)    ( row 10,col 1)    ( row 10,col 2)    ( row 10,col 3)    ( row 10,col 4)
     ( row 11,col 0)    ( row 11,col 1)    ( row 11,col 2)    ( row 11,col 3)    ( row 11,col 4)
     ( row 12,col 0)    ( row 12,col 1)    ( row 12,col 2)    ( row 12,col 3)    ( row 12,col 4)
     ( row 13,col 0)    ( row 13,col 1)    ( row 13,col 2)    ( row 13,col 3)    ( row 13,col 4)
     ( row 14,col 0)    ( row 14,col 1)    ( row 14,col 2)    ( row 14,col 3)    ( row 14,col 4)
     ( row 15,col 0)    ( row 15,col 1)    ( row 15,col 2)    ( row 15,col 3)    ( row 15,col 4)
     ( row 16,col 0)    ( row 16,col 1)    ( row 16,col 2)    ( row 16,col 3)    ( row 16,col 4)
     ( row 17,col 0)    ( row 17,col 1)    ( row 17,col 2)    ( row 17,col 3)    ( row 17,col 4)
     ( row 18,col 0)    ( row 18,col 1)    ( row 18,col 2)    ( row 18,col 3)    ( row 18,col 4)
     ( row 19,col 0)    ( row 19,col 1)    ( row 19,col 2)    ( row 19,col 3)    ( row 19,col 4)
Press any key to continue . . .
```

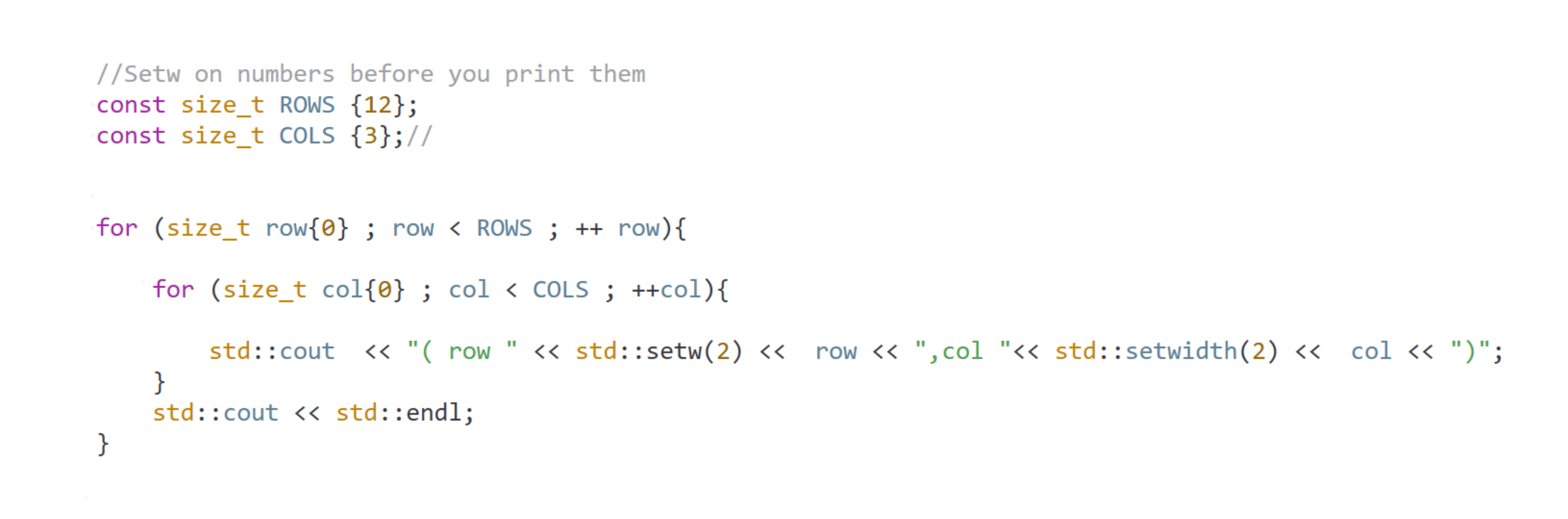
3D



General Thought

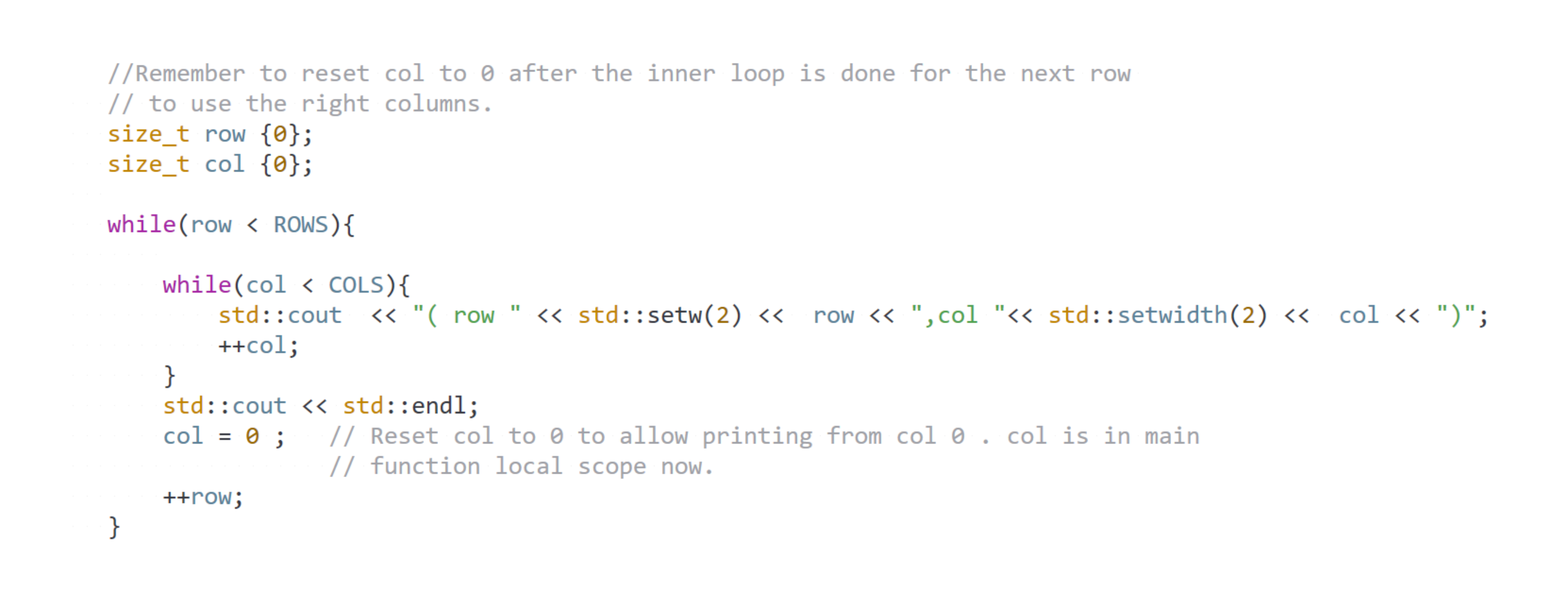
It’s not easy to visually represent data structures with a dimension greater than 3, but now the idea should be building up that the more layers of nests you have in your loops, the more dimensions the data you are looping around could represent.

2D tabular data visualization with nested for loops



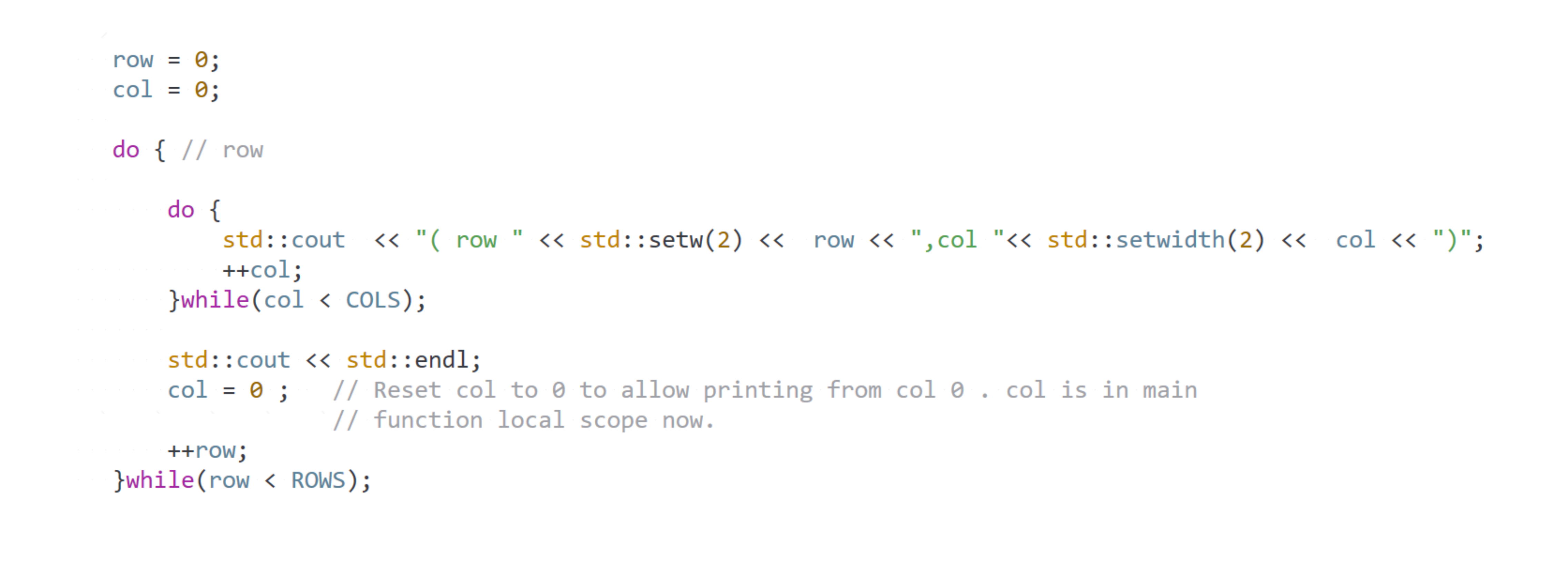
```
//Setw on numbers before you print them
const size_t ROWS {12};
const size_t COLS {3};//


for (size_t row{0} ; row < ROWS ; ++ row){

    for (size_t col{0} ; col < COLS ; ++col){

        std::cout  << "( row " << std::setw(2) <<  row << ",col "<< std::setwidth(2) <<  col << ")";
    }
    std::cout << std::endl;
}
```

2D tabular data visualization with nested while loops



```
//Remember to reset col to 0 after the inner loop is done for the next row
// to use the right columns.
size_t row {0};
size_t col {0};

while(row < ROWS){

    while(col < COLS){
        std::cout  << "( row " << std::setw(2) <<  row << ",col "<< std::setwidth(2) <<  col << ")";
        ++col;
    }
    std::cout << std::endl;
    col = 0 ;   // Reset col to 0 to allow printing from col 0 . col is in main
                // function local scope now.
    ++row;
}
```

2D tabular data visualization with nested do while loops



```
row = 0;
col = 0;

do { // row

    do {
        std::cout  << "( row " << std::setw(2) <<  row << ",col "<< std::setwidth(2) <<  col << ")";
        ++col;
    }while(col < COLS);

    std::cout << std::endl;
    col = 0 ;   // Reset col to 0 to allow printing from col 0 . col is in main
                // function local scope now.
    ++row;
}while(row < ROWS);
```

Range based for loops

Range based for loops aren’t a good fit for visually representing multi dimensional data structures . It’s possible to do that though. We’ll see how to do that when we have more tools in the next chapter

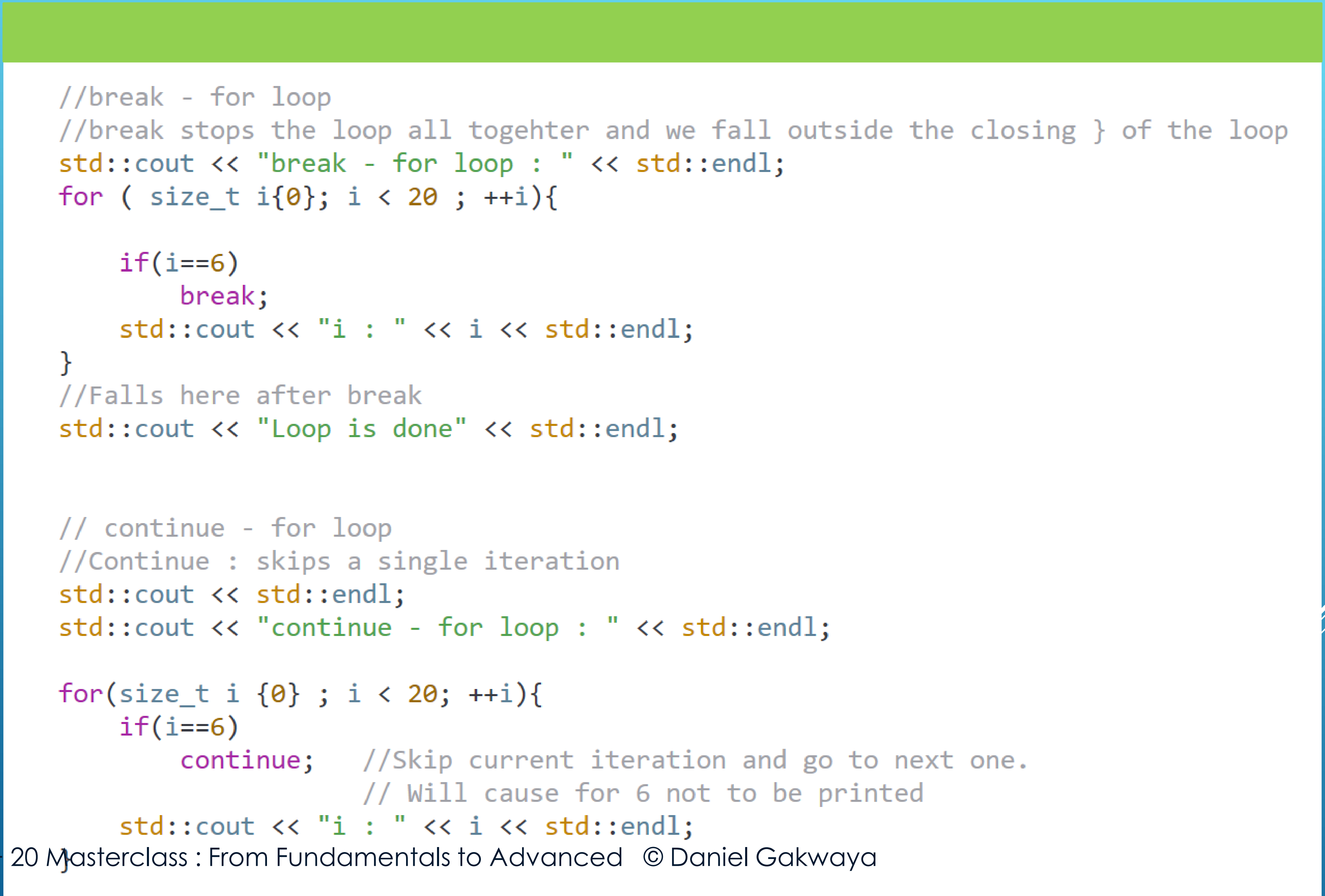
Slide intentionally left empty

break and continue

break

break breaks out of the loop immediately and causes execution of the statement following the loop immediately continue

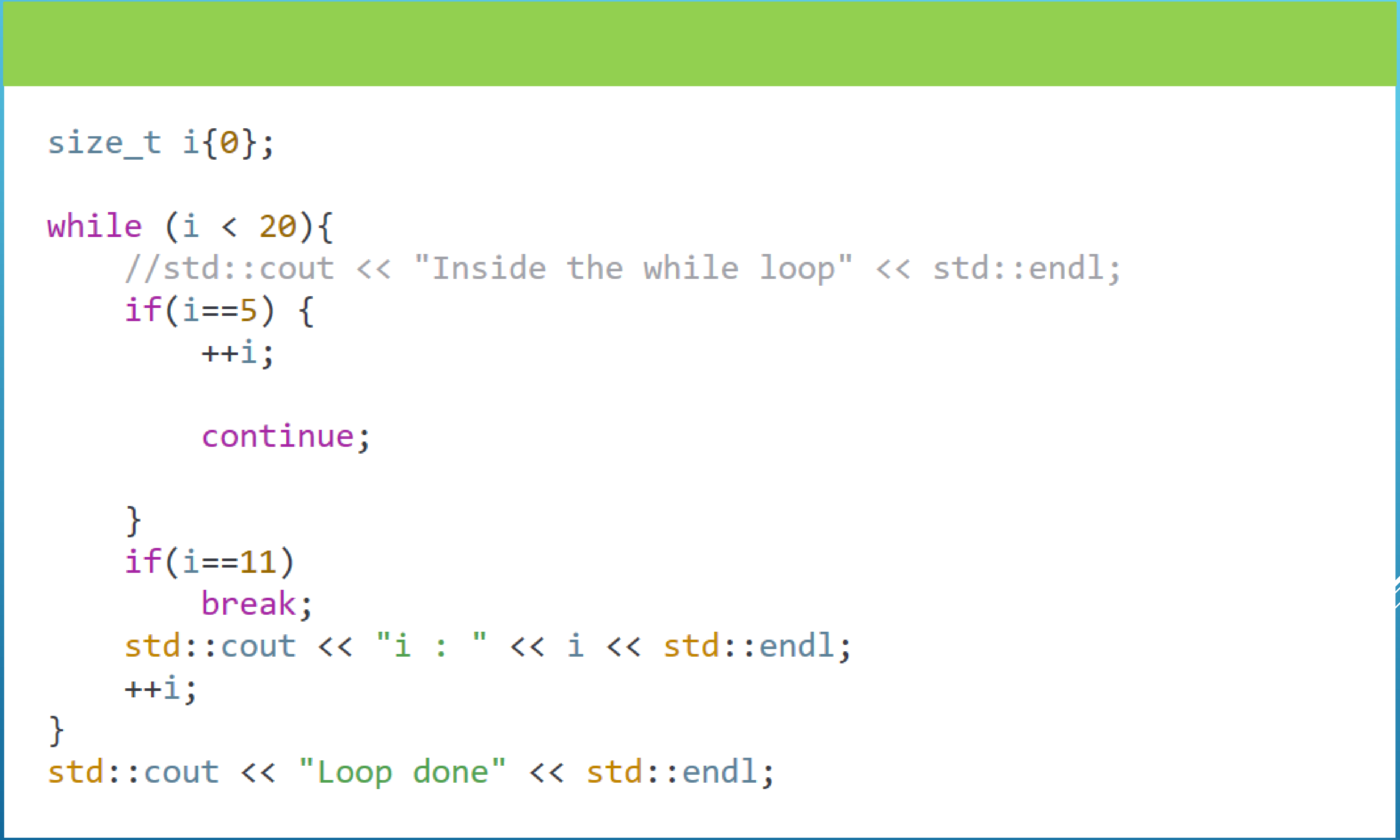
continue allows you to skip a single iteration and move to the next one



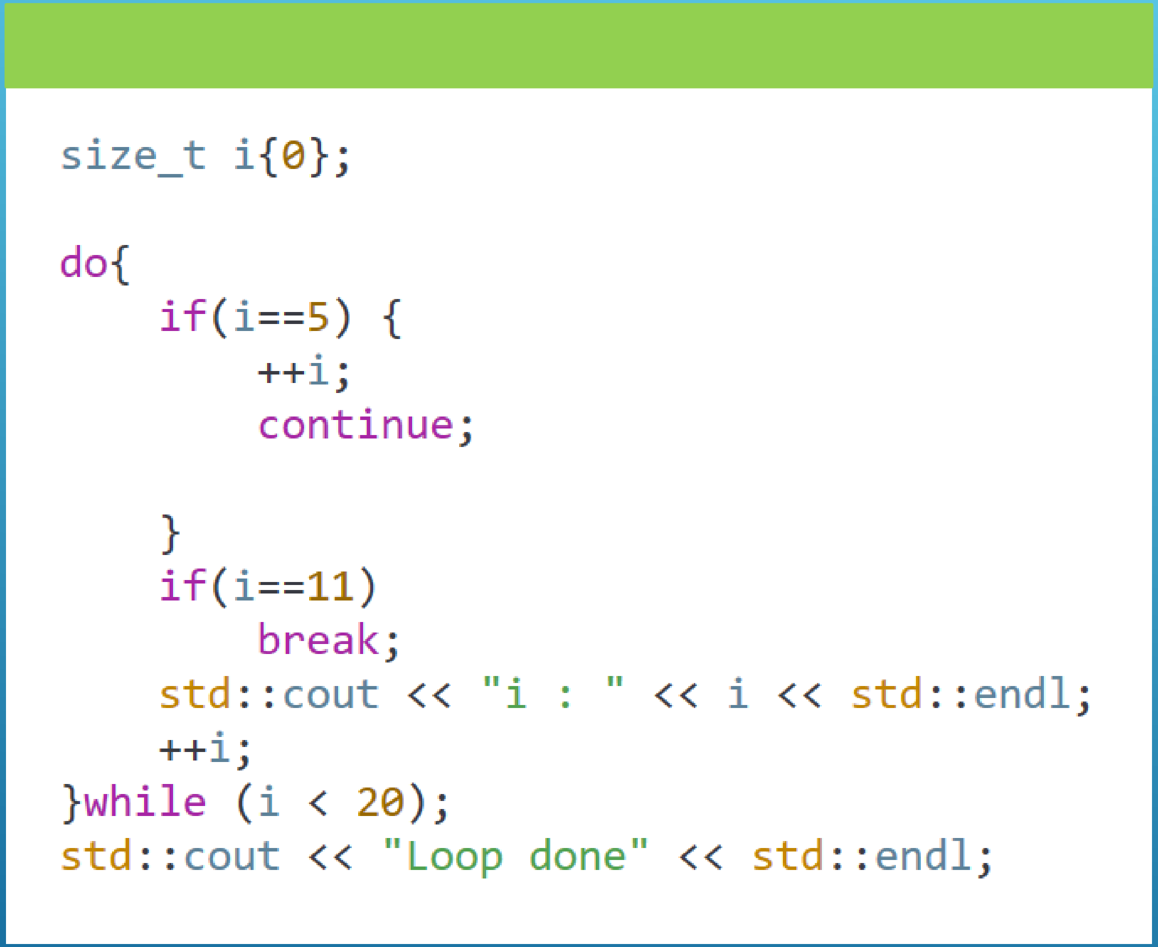
```
//break - for loop
//break stops the loop all togehter and we fall outside the closing } of the loop
std::cout << "break - for loop : " << std::endl;
for ( size_t i{0}; i < 20 ; ++i){

    if(i==6)
        break;
    std::cout << "i : " << i << std::endl;
}
//Falls here after break
std::cout << "Loop is done" << std::endl;


// continue - for loop
//Continue : skips a single iteration
std::cout << std::endl;
std::cout << "continue - for loop : " << std::endl;

for(size_t i {0} ; i < 20; ++i){
    if(i==6)
        continue;   //Skip current iteration and go to next one.
                    // Will cause for 6 not to be printed
    std::cout << "i : " << i << std::endl;
}
```

Danger

continue may cause for the incrementing/ decrementing statement not to be executed and you'll get an infinite loop.



```
size_t i{0};

while (i < 20){
    //std::cout << "Inside the while loop" << std::endl;
    if(i==5) {
        ++i;

        continue;

    }
    if(i==11)
        break;
    std::cout << "i : " << i << std::endl;
    ++i;
}
std::cout << "Loop done" << std::endl;
```



```cpp
size_t i{0};

do{
    if(i==5) {
        ++i;
        continue;

    }
    if(i==11)
        break;
    std::cout << "i : " << i << std::endl;
    ++i;
}while (i < 20);
std::cout << "Loop done" << std::endl;
```

The possibility for infinite loop is not present with continue with for loops because the incrementing/decrementing part is kind of built in. It always executes when control leaves the body of the loop.

Slide intentionally left empty

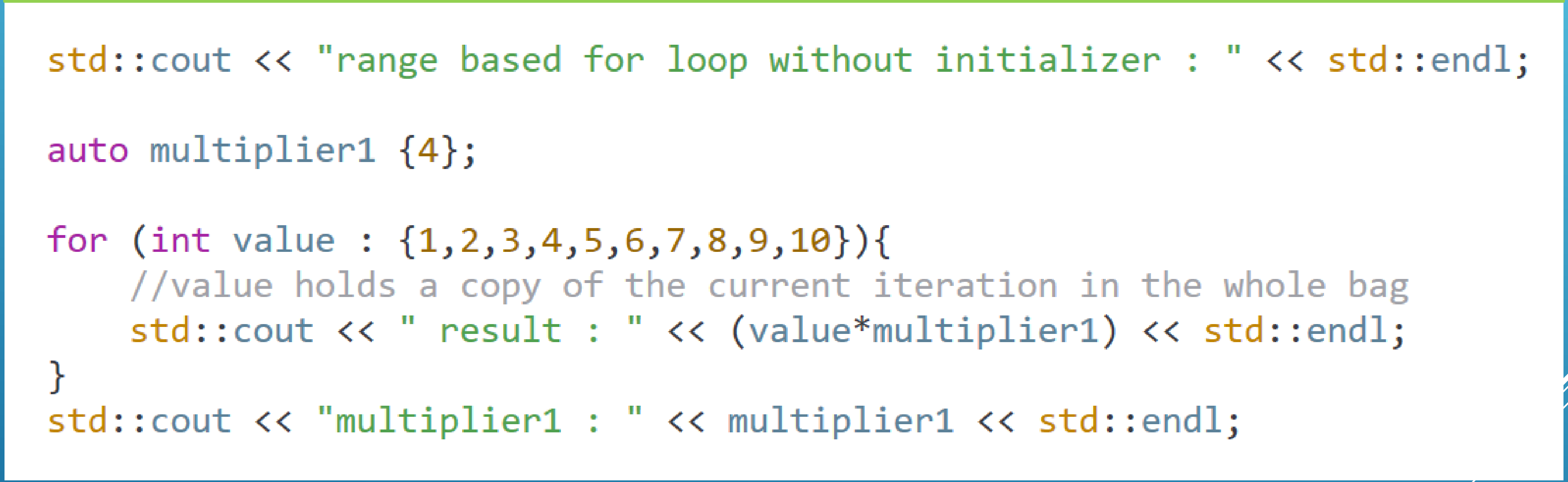
Fixing our Calculator

break	Continue when the operation is not a valid one

Slide intentionally left empty

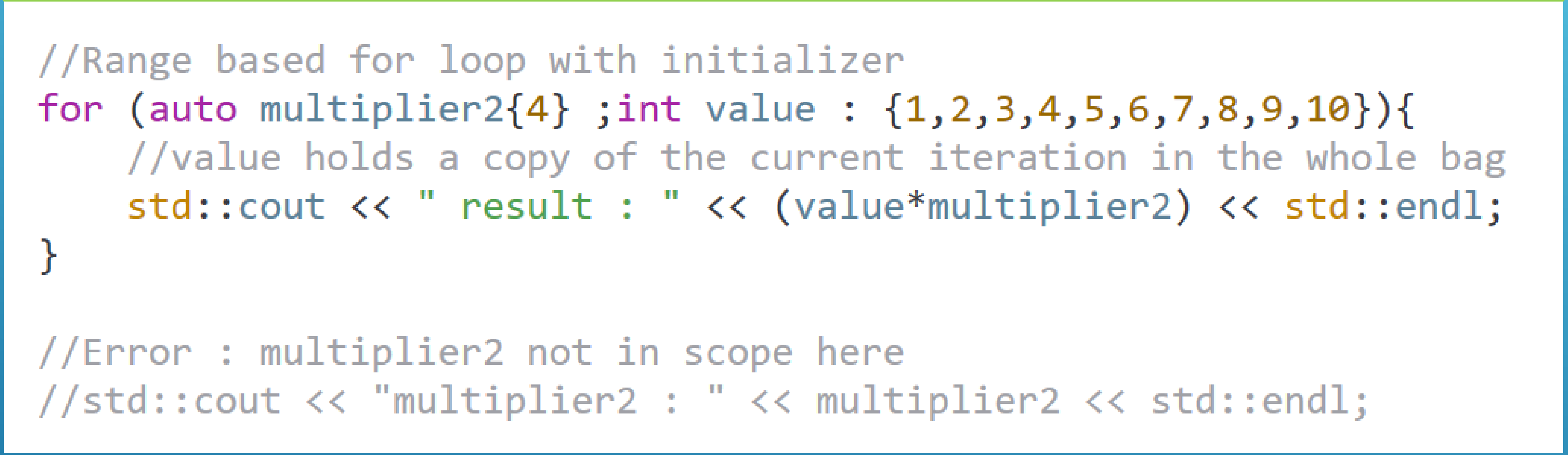
Range based for loop with initializer

Helper variable in outer scope



```cpp
std::cout << "range based for loop without initializer : " << std::endl;

auto multiplier1 {4};

for (int value : {1,2,3,4,5,6,7,8,9,10}){
    //value holds a copy of the current iteration in the whole bag
    std::cout << " result : " << (value*multiplier1) << std::endl;
}
std::cout << "multiplier1 : " << multiplier1 << std::endl;
```

Helper variable in loop scope (C++20)

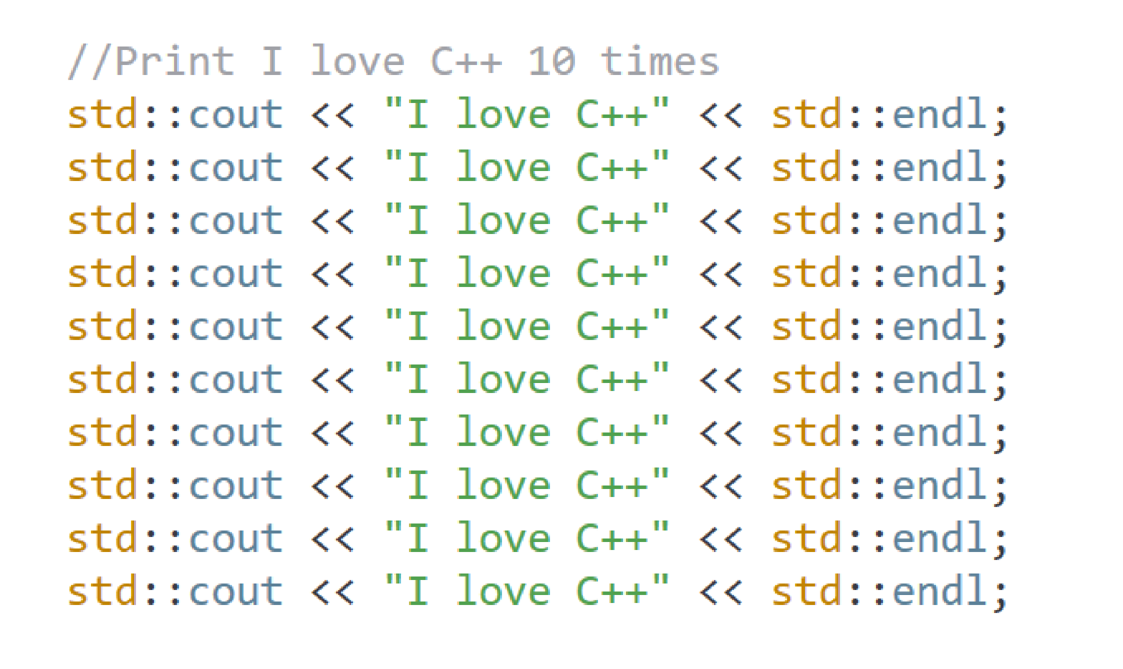


```
//Range based for loop with initializer
for (auto multiplier2{4} ;int value : {1,2,3,4,5,6,7,8,9,10}){
    //value holds a copy of the current iteration in the whole bag
    std::cout << " result : " << (value*multiplier2) << std::endl;
}

//Error : multiplier2 not in scope here
//std::cout << "multiplier2 : " << multiplier2 << std::endl;
```

Slide intentionally left empty

Loops : Summary

Repetitive tasks



```
//Print I love C++ 10 times
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
std::cout << "I love C++" << std::endl;
```

Loops

Pillars of any loop

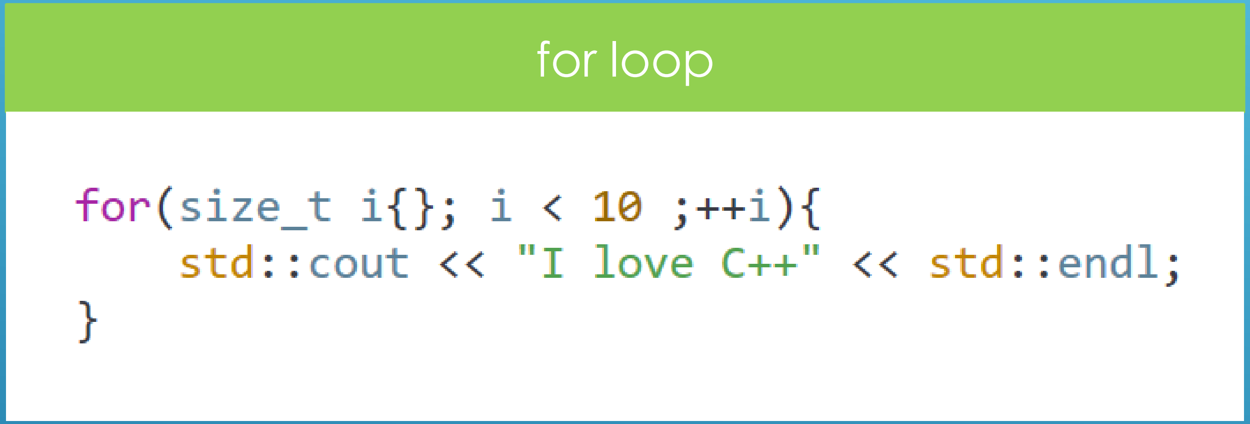
- Iterator
- Starting Point
- Test(controls when the loop stops)
- Increment(Decrement)
- Loop body

For loop

Range based for loop

While loop

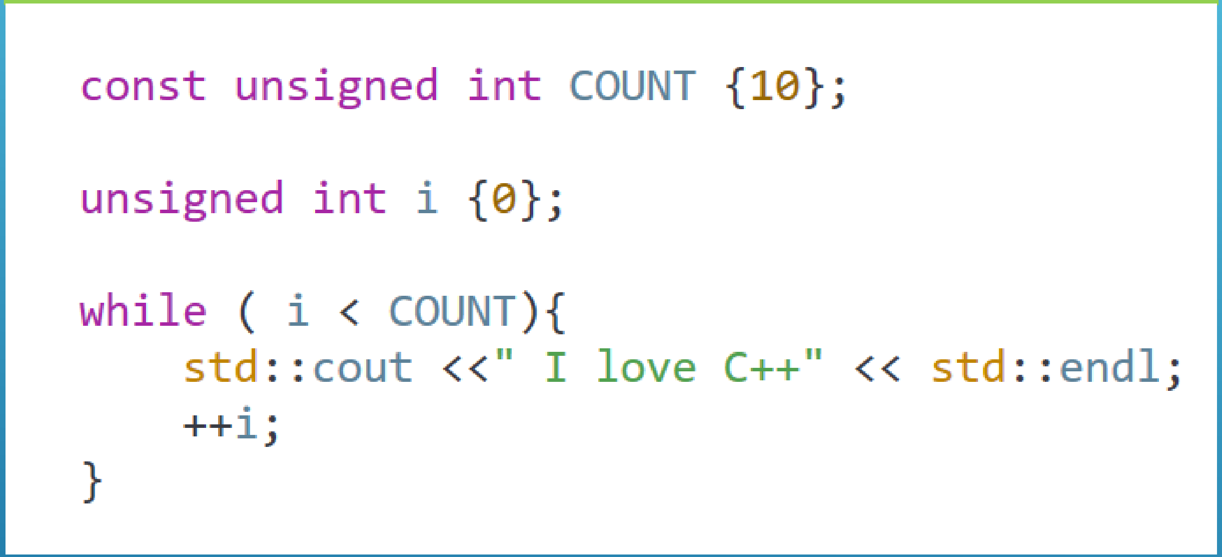
Do while loop



for loop

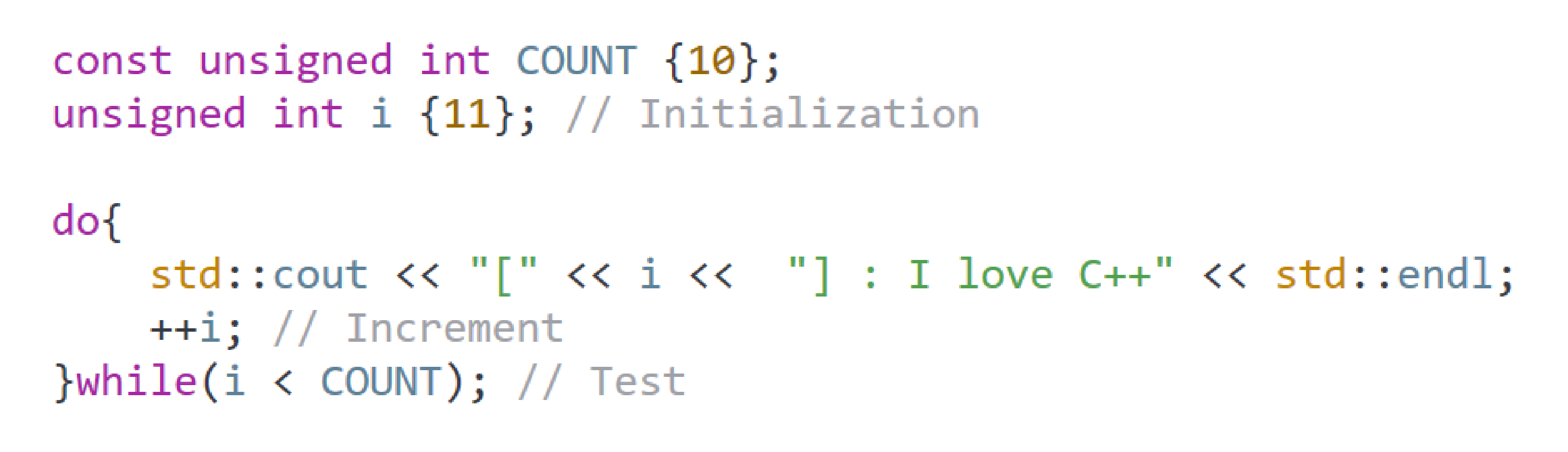
```
for(size_t i{}; i < 10 ;++i){
    std::cout << "I love C++" << std::endl;
}
```

while loop



```
const unsigned int COUNT {10};

unsigned int i {0};

while ( i < COUNT){
    std::cout <<" I love C++" << std::endl;
    ++i;
}
```

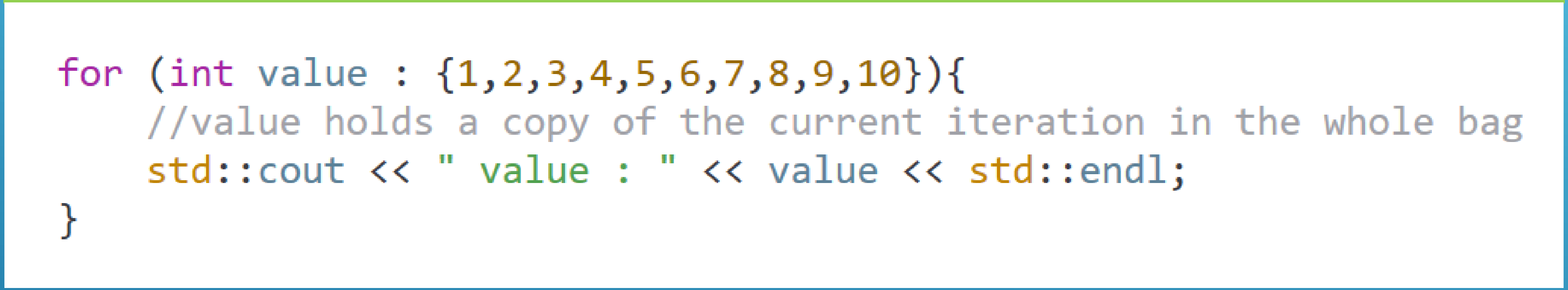
Loop with output



```cpp
const unsigned int COUNT {10};
unsigned int i {11}; // Initialization

do{
    std::cout << "[" << i <<  "] : I love C++" << std::endl;
    ++i; // Increment
}while(i < COUNT); // Test
```

size_t

Range based for loop



```
for (int value : {1,2,3,4,5,6,7,8,9,10}){
    //value holds a copy of the current iteration in the whole bag
    std::cout << " value : " << value << std::endl;
}
```

Infinite loops

Loops with std::cout

Decrementing loops

Nested loops

break and continue

Comma operator