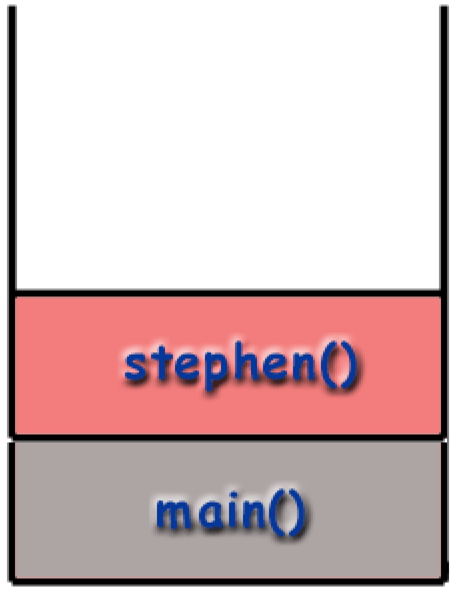
Understanding How Functions Work

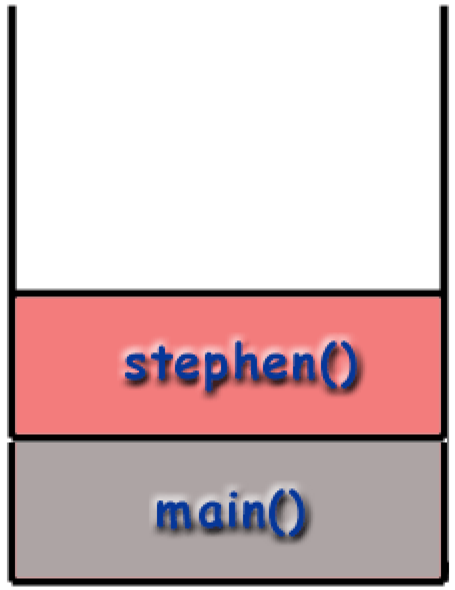
Function Scope

- Each function has its own separate memory space isolated from all the other functions.
- Variables declared in the body of a function are stored in this memory space.
- These variables are called local variables.



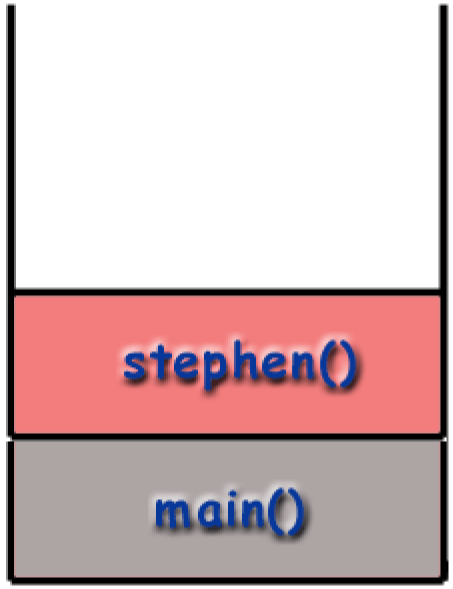
Function Scope

- Local variables can only be accessed by code written inside the same function body where the variables were declared.
- This means local variables are only visible (in scope) inside the function where they were declared.
- Example: variables declared in the function, `stephen()`, can't be used in `main()` because they aren't in scope in `main()`.



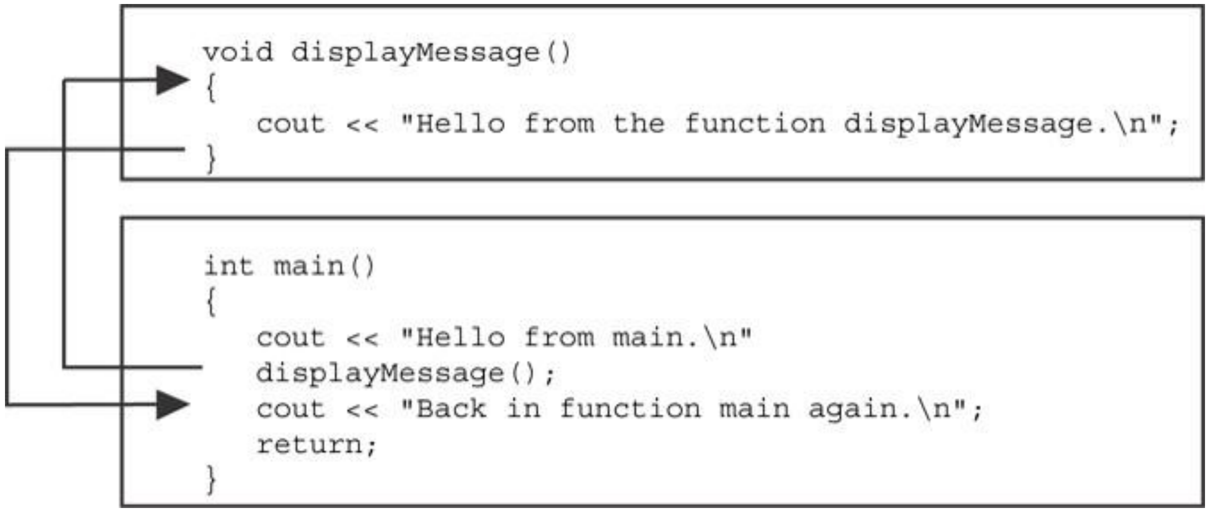
Function Scope

- Memory reserved for a function exists only while the function is executing.
- Once the function finishes executing, the memory reserved for it disappears.
- All variables declared inside the function will disappear once the function finishes executing.
- Example: once the function, `stephen()`, finishes executing, the values stored in the variables declared there disappear.



Flow of Control

- When a program begins executing, program control is inside the `main()` function.
- Once a function is called from `main()`, control leaves `main()` and is inside the function.

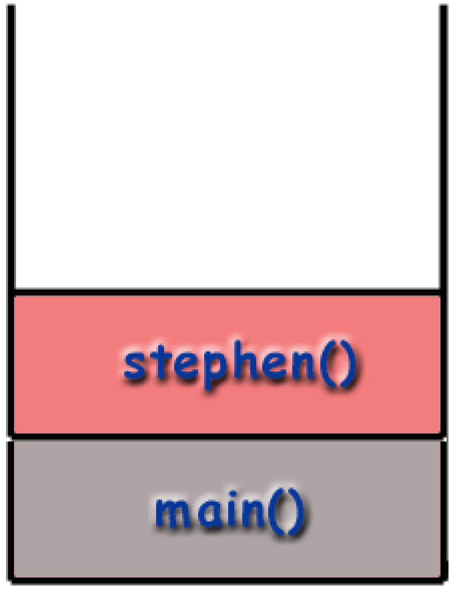


```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}
```

```
int main()
{
   cout << "Hello from main.\n"
   displayMessage();
   cout << "Back in function main again.\n";
   return;
}
```

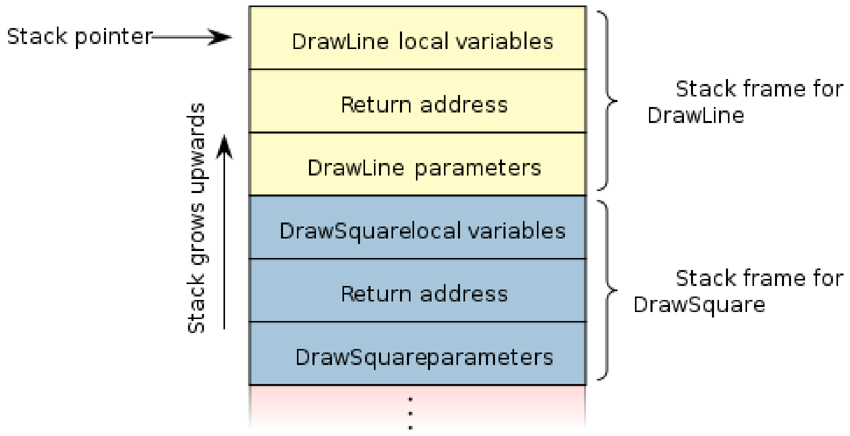
Flow of Program Execution

- A mechanism exists to keep track of program control and to manage the memory reserved for each function.
- This mechanism is called the call stack.
- While a function is executing, its memory is stored on the call stack.
- When a function finishes executing, its memory is removed from the call stack.



Call Stack

- Keeps track of flow of program execution.
- Each function has its own memory space (frame).
- Most recently called function’s frame is on top.
- When function finishes executing, its frame is “popped” off the top of the stack (memory deleted) and control is returned to the calling function.



Return Values

- When control is returned to the calling function, the value stored in one of the function's local variables can also be returned.
- That value must be saved in a variable declared inside the calling function in order to use it again.
- Otherwise, the value will be lost once the frame is popped off the stack.

```
int add(int a, int b)
{
  int  total = a + b;
  return total;
}

int main()
{
 int num1 = 2;
 int num2 = 4;
 int sum = add(num1, num2);
 cout << sum << endl;
}
```

Calling function

Pass Values

- Because each function has its own memory space, values stored in local variables must be copied into the memory space of another function if that function needs the values to complete its task.
- The called function must declare variables (parameters) to store the passed values.

Parameters

```
int add(int a, int b)
{
  int  total = a + b;
  return total;
}

int main()
{
 int num1 = 2;
 int num2 = 4;
 int sum = add(num1, num2);
 cout << sum << endl;
}
```

Pass Values

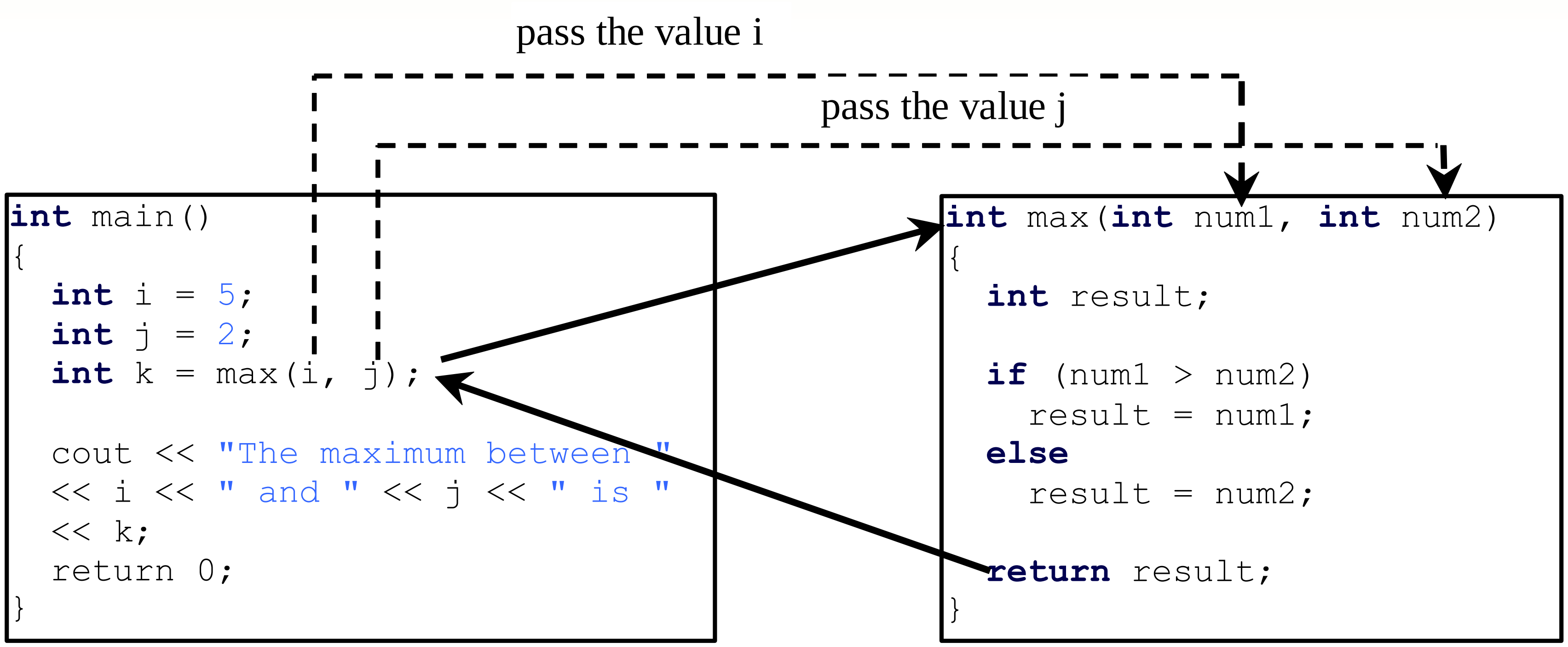
- In the example given, the values stored in the local variables, `num1` and `num2`, are copied into the parameters, `a` and `b`.
- Both `a` and `num1` contain 2 because the first argument is always copied into the first parameter.
- Both `b` and `num2` contain 4 because the second argument is copied into the second parameter.

Parameters

```
int add(int a, int b)
{
   int  total = a + b;
   return total;
}

int main()
{
  int num1 = 2;
  int num2 = 4;
  int sum = add(num1, num2);
  cout << sum << endl;
}
```

Arguments

Calling Functions – Stack Controls Flow of Execution



```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
  << i << " and " << j << " is "
  << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

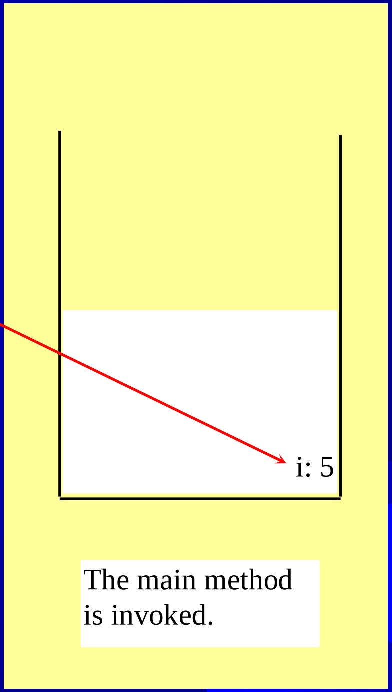
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

i is declared and initialized



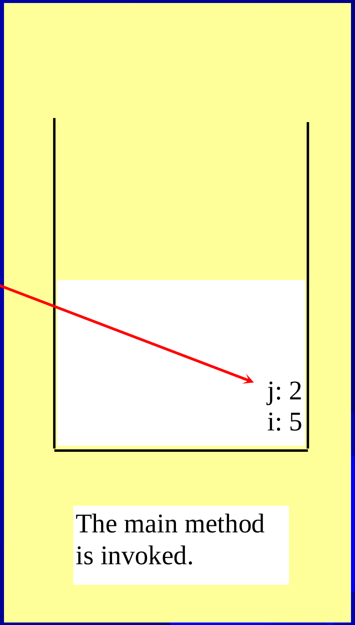
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

j is declared and initialized



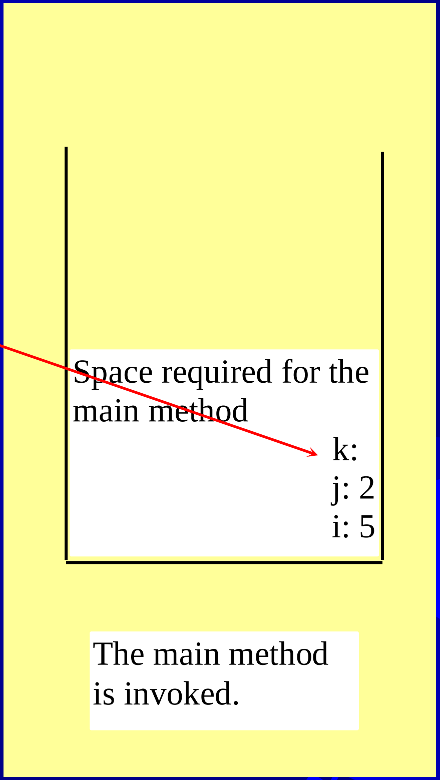
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Declare k


Space required for the main method
k:
j: 2
i: 5
The main method is invoked.

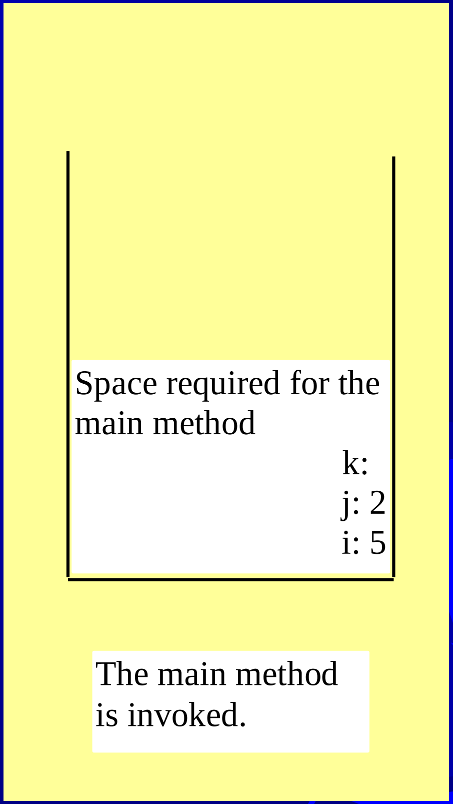
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Invoke max(i, j)


Space required for the main method
k:
j: 2
i: 5
The main method is invoked.

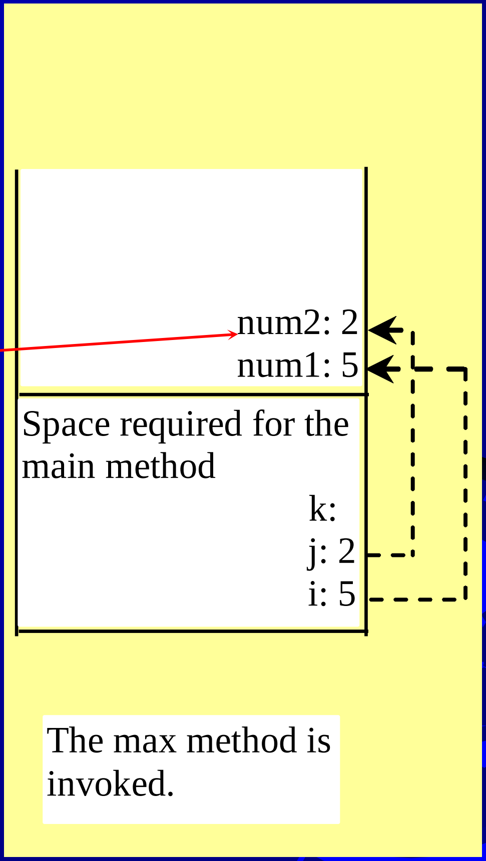
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

pass the values of i and j to num1 and num2


num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5
The max method is invoked.

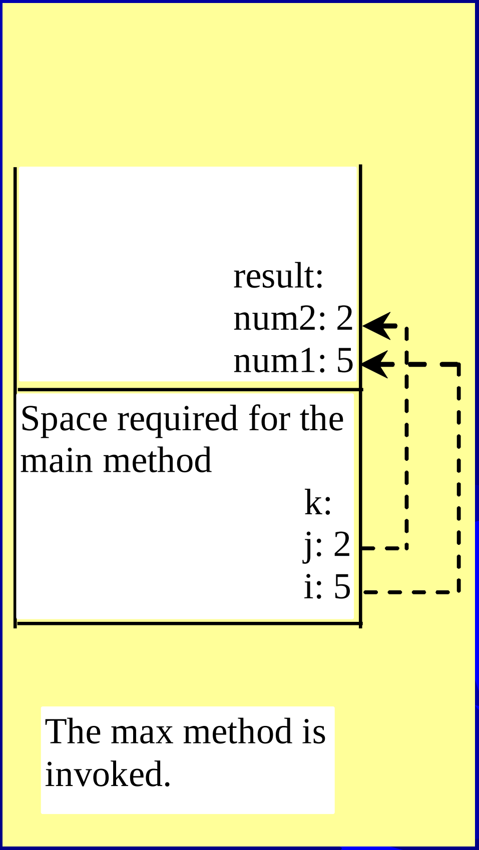
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Declare result



The max method is invoked.

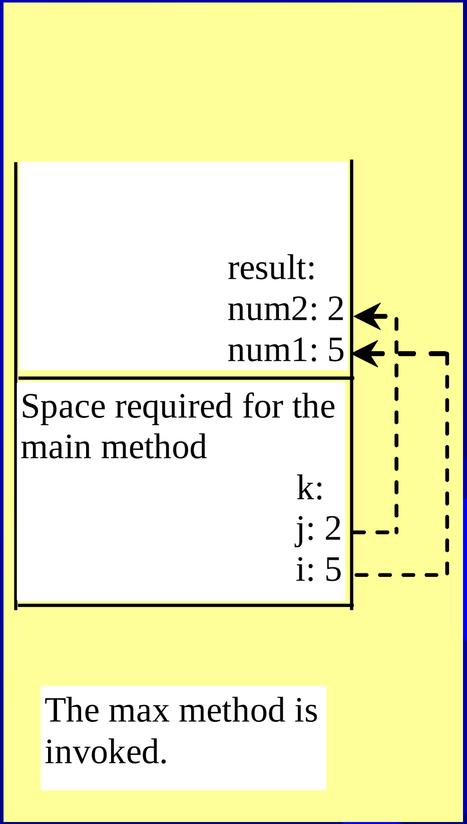
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

(num1 > num2) is true



The max method is invoked.

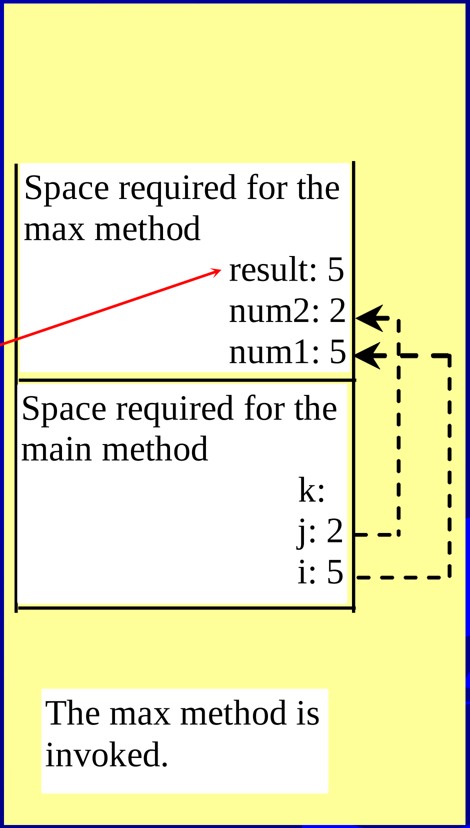
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Assign num1 to result


Space required for the max method
result: 5
num2: 2
num1: 5
Space required for the main method
k:
j: 2
i: 5
The max method is invoked.

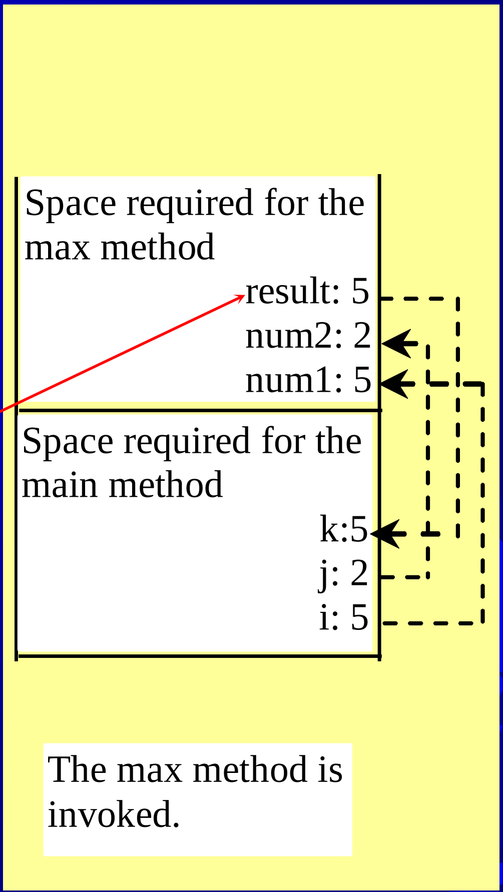
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is "
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Return result and assign it to k



Space required for the max method
result: 5
num2: 2
num1: 5
Space required for the main method
k:5
j: 2
i: 5

The max method is invoked.

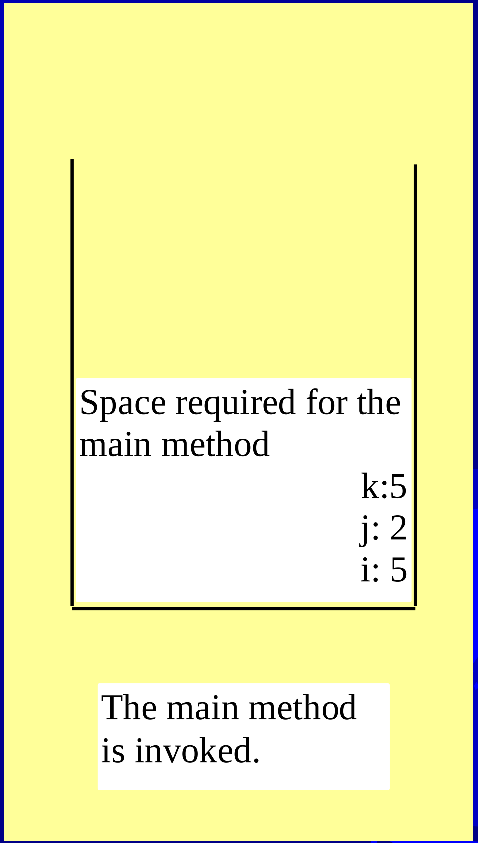
animation

Trace Call Stack

```
int main()
{
  int i = 5;
  int j = 2;
  int k = max(i, j);

  cout << "The maximum between "
    << i << " and " << j << " is
    << k;
  return 0;
}
```

```
int max(int num1, int num2)
{
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Execute print statement



Space required for the main method
k:5
j: 2
i: 5
The main method is invoked.