

A Simple Stack Example

July 20, 2009

Reading These Slides

Line 19: Call function 'a'

create formal param 'b' of 'a'

Descriptive text with the current line number is provided at the top of each slide. One C++ statement may actually generate multiple CPU instructions: these instructions are displayed in a second descriptive line which is a lighter shade of blue.

Line 19: Call function 'a'

create formal param 'b' of 'a'

$\langle \text{OS} \rangle \implies \text{main}$

Below the descriptive text is a call graph of the application, it begins with just OS, which is the operating system (Windows, Linux, ...). The operating system calls `main()`, and `main()` will then call other functions. The call graph shows the nesting of function calls that matches the application stack.

Line 19: Call function 'a'

create formal param 'b' of 'a'

<OS> \implies main

To the right, <== marks the the C++ statement that is currently executing. Some C++ statements perform multiple CPU instructions, so the marker doesn't necessarily move with every stack change.

```
1  /**
2  * A test file for the stack
3  * visualization framework
4  */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     int x(3);
19     a(x); // <==
20     cout << x << endl;
21     return 0;
22 }
```


Line 19: Call function 'a'

create formal param 'b' of 'a'

Critical to understanding these slides is knowing that the call graph, code listing (with `<==` line indicator), and the program memory diagram (the stack and the heap), all represent the program state **AFTER** the program step described in the title has taken place.

That's a long sentence, read it again. Be sure you understand it.

It means that when elements are **CREATED** in memory the slides are easy to interpret (because the newly created element in memory can be seen on the slide!).

But, when elements are **DESTROYED** in memory, you may need to compare the program memory diagram with the figure *one slide back*, to really see the effect.

A Simple Example

The following is an example of how C++ (and programs in general) keep track of function calls, parameters passed to functions by value, and the local variables of functions.

OS calls main()

<OS>



```
1 /**
2  * A test file for the stack
3  * visualization framework
4  */
5 #include <iostream>
6 using namespace std;
7
8 void a( double b )
9 {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0;
22 }
```

OS calls main()

create call frame for main

<OS>



```
1  /**
2  * A test file for the stack
3  * visualization framework
4  */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0;
22 }
```

Line 16: Function main entry

<OS> \implies main

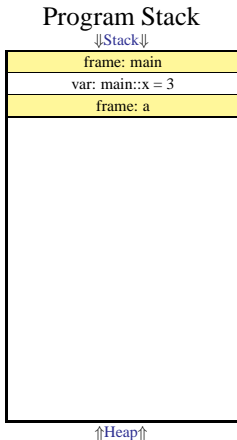


```
1 /**
2  * A test file for the stack
3  * visualization framework
4  */
5 #include <iostream>
6 using namespace std;
7
8 void a( double b )
9 {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main() // <==
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0;
22 }
```


Line 19: Call function a

create call frame for a

<OS> \implies main

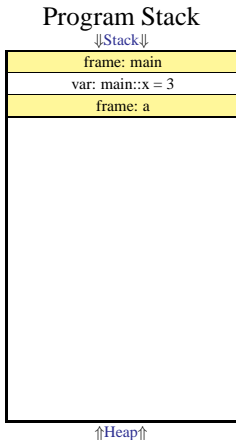


```
1  /**
2   * A test file for the stack
3   * visualization framework
4   */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x); // <==
20     cout << x << endl;
21     return 0;
22 }
```


Line 13: Return to main

discard a parameter b

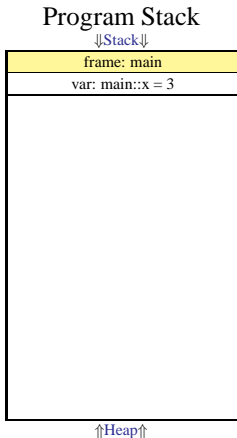
<OS> \implies main \implies a



```
1  /**
2   * A test file for the stack
3   * visualization framework
4   */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return; // <==
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0;
22 }
```


Line 19: Return from a

<OS> \implies main



```
1 /**
2  * A test file for the stack
3  * visualization framework
4  */
5 #include <iostream>
6 using namespace std;
7
8 void a( double b )
9 {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x); // <==
20     cout << x << endl;
21     return 0;
22 }
```


Line 21: Return to OS

discard the auto (local) variable `main::x`

<OS> \implies main

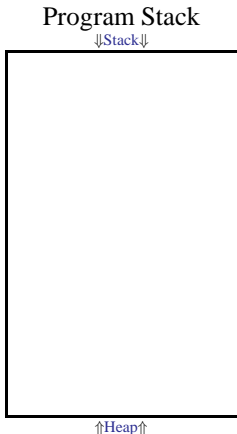


```
1  /**
2   * A test file for the stack
3   * visualization framework
4   */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0; // <==
22 }
```

Line 21: Return to OS

discard call frame, return to caller

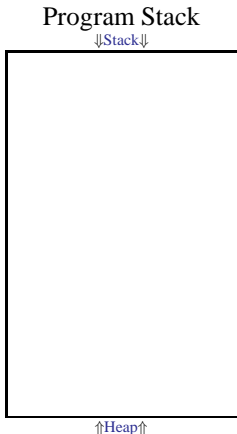
<OS> \implies main



```
1  /**
2   * A test file for the stack
3   * visualization framework
4   */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0; // <==
22 }
```

Return from main

<OS>



```
1  /**
2   * A test file for the stack
3   * visualization framework
4   */
5  #include <iostream>
6  using namespace std;
7
8  void a( double b )
9  {
10     double c;
11     c = b + 1.010010001;
12     cout << c << endl;
13     return;
14 }
15
16 int main()
17 {
18     double x(3);
19     a(x);
20     cout << x << endl;
21     return 0;
22 }
```

finis