Variables, Functions & Parameter Passing

CSci 588
Fall 2013

Variables

Variables consist of a name/type pair. The name can be anything (starting with '_' or a letter), while the type refers to some kind of data.

Variables

Within a function, variables reside in a set of *registers* for that function. These are temporary spots in memory (think on CPU memory), and each function will have it's own set of registers.

When you create an array or an object with the 'new' keyword, it actually allocates persistent memory (think RAM) that doesn't go away until you delete it with the 'delete' keyword.

A Simple Example

A Simple Example

```
#include <iostream>

int main(int number_of_arguments, char **arguments) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

A Simple Example

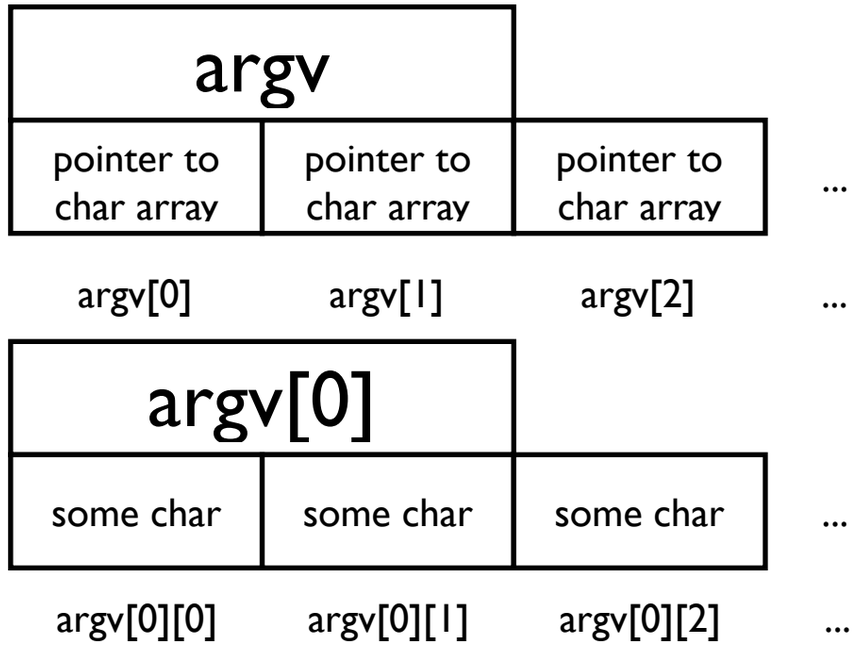
```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

c++ will create an array of references to arrays of characters (an array of characters is an old style c string). These are in memory. c++ also makes two registers, an int which we're calling 'argc' and a pointer to the array of pointers to arrays of characters which we're calling 'argv'.

Registers:

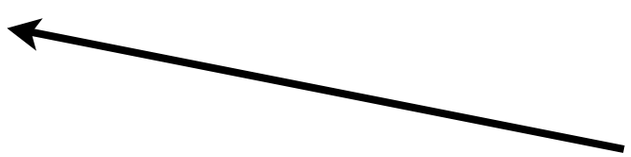
argc	argv
some number	pointer to memory

Memory:



A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

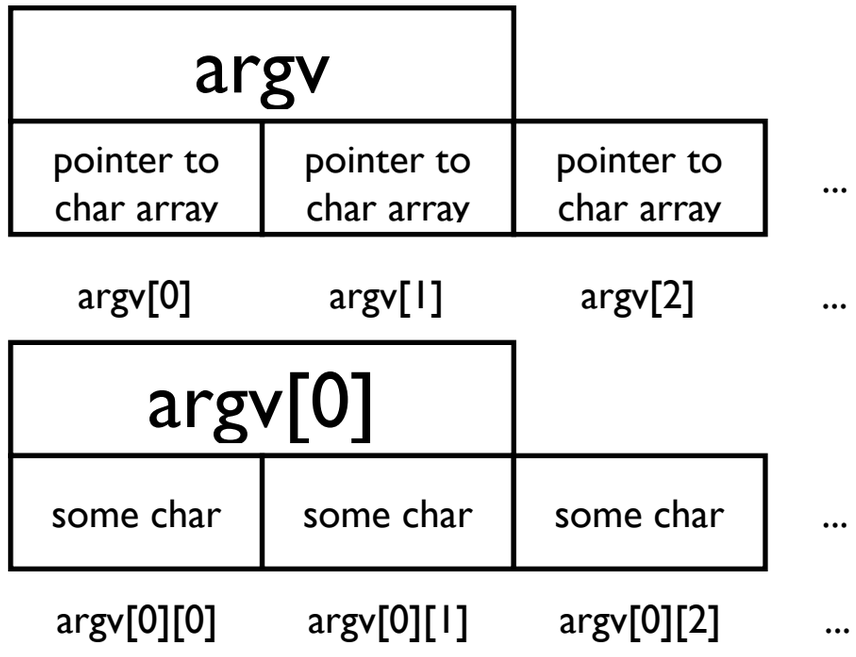


'argc' is short for argument count (it's the number of command line arguments), and 'argv' is short for argument vector (it's the number of arguments passed by the command line).

Registers:

argc	argv
some number	pointer to memory

Memory:



A Simple Example

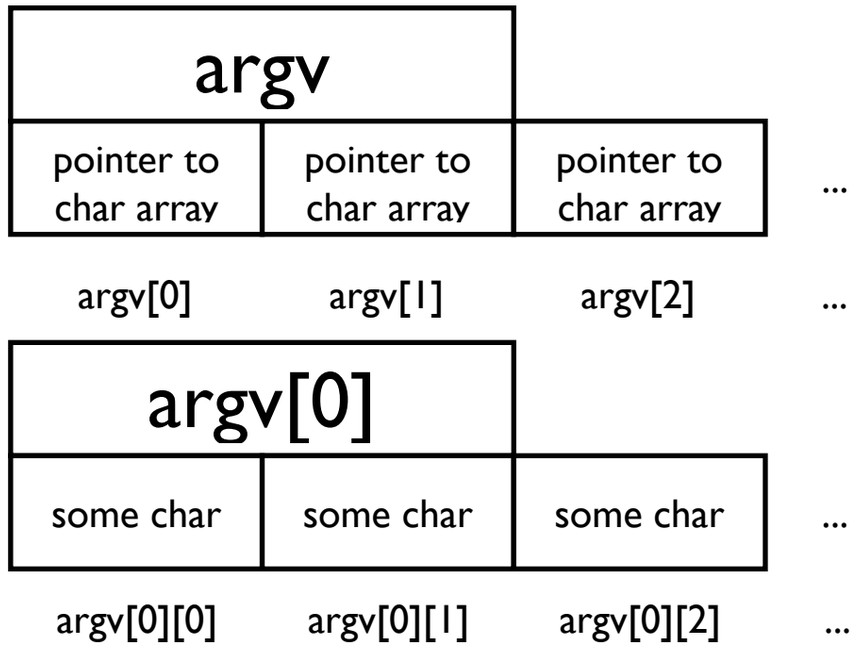
```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

Note that there will be an array in memory for each command line argument, argv[0], argv[1], argv[2], ... and so on. But since we're not using these I'm going to remove them from the rest of the slides.

Registers:

argc	argv
some number	pointer to memory

Memory:



A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

This statement will create two registers with enough space in each for an int.

Registers:

argc	argv	x	y
some number	pointer to memory		

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

This will set the data at the register for ‘x’ to 5.

Registers:

argc	argv	x	y
some number	pointer to memory	5	

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

This will set the data at the register for ‘y’ to 3.

Registers:

argc	argv	x	y
some number	pointer to memory	5	3

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

This will create the register for the ‘temp’ variable and assign it’s data to the data in the ‘x’ variables register in one line.

Registers:

argc	argv	x	y	temp
some number	pointer to memory	5	3	5

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

Note that the data at the x register does not change, it's only copied over to the temp register.

Registers:

argc	argv	x	y	temp
some number	pointer to memory	5	3	5

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

This will set the data at the ‘x’ register to the data at the ‘y’ register. Again, the data at the ‘y’ register does not change, nor does the data at the ‘temp’ register.

Registers:

argc	argv	x	y	temp
some number	pointer to memory	3	3	5

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

This will set the data at the ‘y’ register to the data at the ‘temp’ register. ‘temp’ and ‘x’ don’t change, for the same reasons as before. The data is just copied into the register.

Registers:

argc	argv	x	y	temp
some number	pointer to memory	3	5	5

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

When the program finishes, the data for the ‘x’ and ‘y’ variables has been swapped.

Registers:

argc	argv	x	y	temp
some number	pointer to memory	3	5	5

Memory:

A Simple Example

```
#include <iostream>

int main(int argc, char **argv) {
   int x, y;
   x = 5;
   y = 3;

   int temp = x;
   x = y;
   y = temp;
}
```

After the function completes, the registers are removed. Regsiters only live for the duration of the function they are created in, and can only be accessed from within the function they were created in.

Registers:

Memory:

An Example with Methods

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

Remember, the program always starts with the main function.

Also, we'll be ignoring argc and argv again (since we aren't using them).

Registers for 'main':

Call Stack:

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

This line does the same thing as the first three lines of the previous program's main method. It creates two variables 'x' and 'y', along with a register for each large enough to hold an int value.

It then sets the value at x's register to 5 and y's register to 3.

Registers for 'main':

x	y
5	3

Call Stack:

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

Now things get tricky.

We'll make another register called largest, again with enough space for an int.

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

But what do we set largest to?

Now we need to deal with the call stack. The *flow* of the program leaves the main function and goes into the max function.

All the registers of main are put on hold, waiting for the max function to *return*.

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

So we put line 14 of main on the call stack, and then the flow of our program moves into the max function.

Registers for 'main':

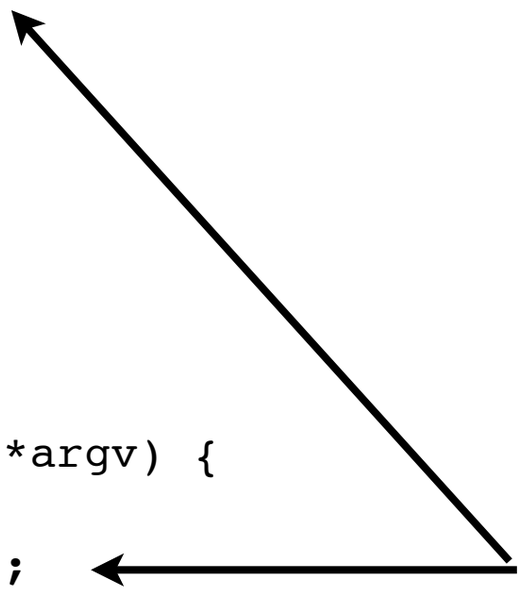
x	y	largest
5	3	?

Call Stack:

main : 14

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



When the flow of the program goes into the new function, the new function has registers made for each of it's arguments, n1 and n2 (just like in the previous example c++ made registers for argc and argv).

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

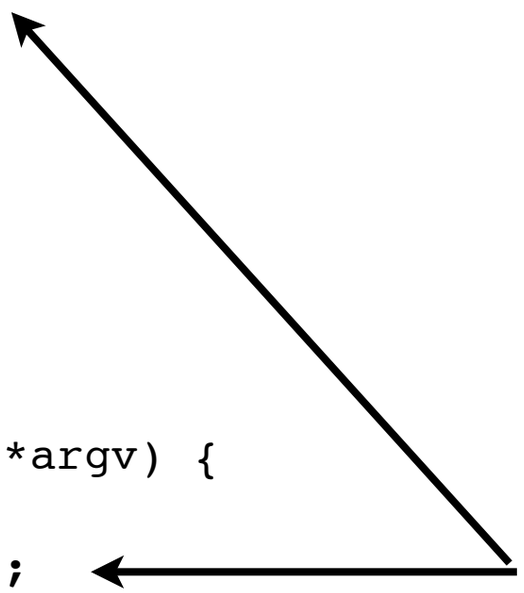
main : 14

Registers for 'max':

n1	n2
?	?

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



Note that the type of the 'x' variable is an int, and the type of 'n1' is also an int -- these parameters match. The type of 'y' is an int, and the type of 'n2' is also an int, so they match as well.

When you call a function (the function is called on line 14), the types arguments you pass into it must match the types in the function declaration (the function is declared on line 4).

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

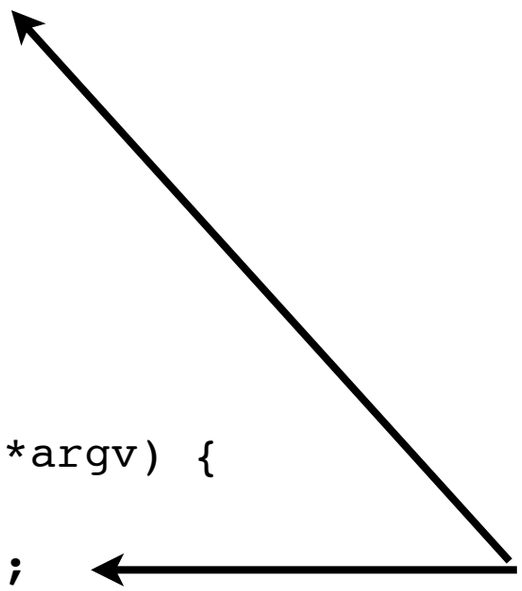
main : 14

Registers for 'max':

n1	n2
?	?

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



When you call the function, it sets the values of the registers of the variables in the function declaration to the values of the variables passed into the function.

In this case, the value of x gets copied into n1's register, and the value of y gets copied into n2's register.

Registers for 'main':

x	y	largest
5	3	?

Registers for 'max':

n1	n2
5	3

Call Stack:

main : 14

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

Now the program has finished all the initialization to call the max function and the program starts going from line 04, the beginning of the max function.

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

main : 14
max : 04

Registers for 'max':

n1	n2
5	3

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

This if statement now compares the values of n1 and n2. If n1 > n2 (n1 is greater than n2), the flow of the program moves into the proceeding block, if not, it moves into the block of code after the else statement. (A block of code is the code between {}s).

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

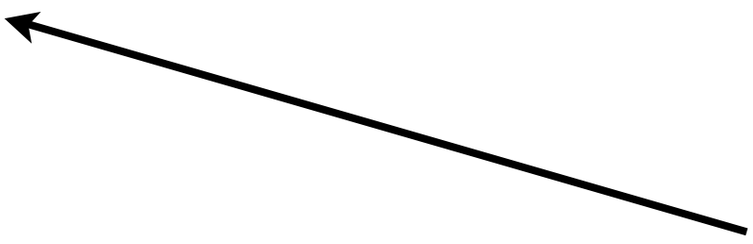
main : 14
max : 05

Registers for 'max':

n1	n2
5	3

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



Since n1 > n2, we move to line 06 (if it had not been, we would have moved to line 08).

Registers for 'main':

x	y	largest
5	3	?

Call Stack:

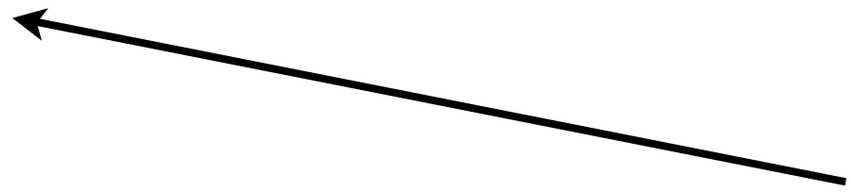
main : 14
max : 05

Registers for 'max':

n1	n2
5	3

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



Now we reach a return statement. A return statement automatically leaves a function, and if a register was waiting for the value of that function; (in this case, the 'largest' register was), that register gets set to the value after the return.

Note that the return type of a function(max returns an int, as declared on line 4), must match the data type of the register it the return value of the function is being assigned to.

Registers for 'main':

x	y	largest
5	3	?

Registers for 'max':

n1	n2
5	3

Call Stack:

main : 14
max : 06

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

So this statement returns the value of the 'n1' register, and assigns the value of the 'largest' register to it.

The flow of the program immediately leaves the max function, it is removed from the call stack and it's registers are removed as well.

Registers for 'main':

x	y	largest
5	3	**5**

Call Stack:

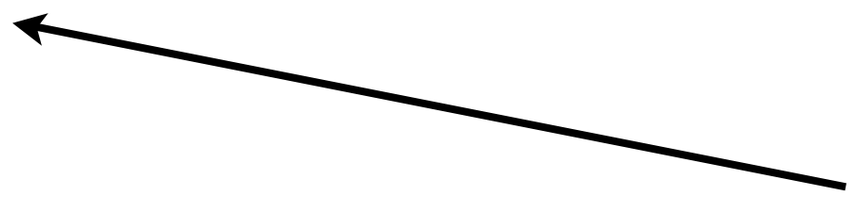
main : 14
max : 06

Registers for 'max':

n1	n2
5	3

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



So this statement returns the value of the ‘n1’ register, and assigns the value of the ‘largest’ register to it.

The flow of the program immediately leaves the max function, it is removed from the call stack and it’s registers are removed as well.

Registers for ‘main’:

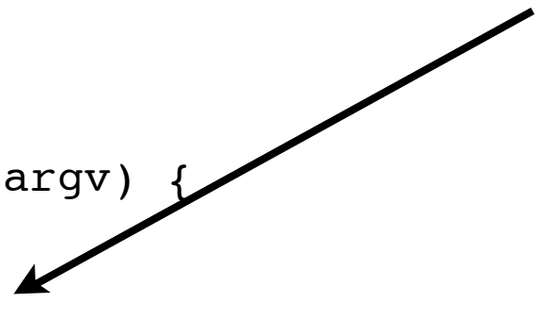
x	y	largest
5	3	**5**

Call Stack:

main : 14

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```



We now go back to line 14 of the main function (which was saved on the call stack), and as the register for largest has been set we can continue in the main function.

Registers for ‘main’:

x	y	largest
5	3	5

Call Stack:

main : 14

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

Line 15 will print out the value at the largest register to the screen.

Registers for ‘main’:

x	y	largest
5	3	5

Call Stack:

main : 15

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

When the main function completes, it is popped off the call stack, the registers are removed and the program is over.

Registers for ‘main’:

x	y	largest
5	3	5

Call Stack:

main : 15

An Example with Methods

```
01 #include <iostream>
02 using namespace std;
03
04 int max(int n1, int n2) {
05   if (n1 > n2) {
06     return n1;
07   } else {
08     return n2;
09   }
10 }
11
12 int main(int argc, char **argv) {
13   int x = 5, y = 3;
14   int largest = max(x, y);
15   cout << largest << endl;
16 }
```

When the main function completes, it is popped off the call stack, the registers are removed and the program is over.

Call Stack:

Another Example with Methods

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Again, your program always starts at the main function.

And again, we're going to ignore the registers and memory for argc and argv because we're not using them.

Registers for 'main':

Call Stack:

main : 13

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Just as before, we make two registers, one for 'x' and one for 'y', each large enough to hold an int.

The x register is assigned a 5 and the y register is assigned a 3.

Registers for 'main':

x	y
5	3

Call Stack:

main : 14

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
15   swap_right(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17 }
```

We don't have a register waiting for a return value, but we're still calling the function.

Just like with the max function, we add swap_wrong to the call stack, create the registers for it's arguments (n1 and n2) and assign their values to x and y.

Registers for 'main':

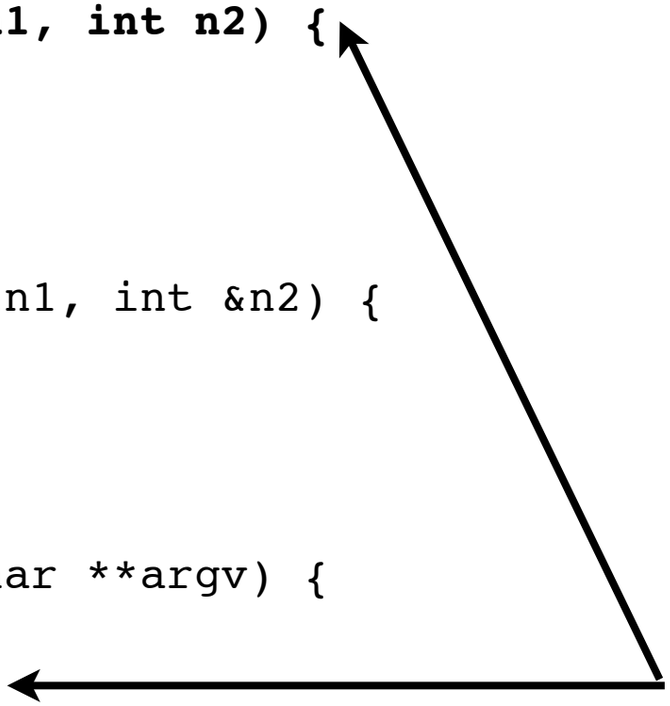
x	y
5	3

Call Stack:

main : 15

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```



We don't have a register waiting for a return value, but we're still calling the function.

Just like with the max function, we add swap_wrong to the call stack, create the registers for it's arguments (n1 and n2) and assign their values to x and y.

Registers for 'main':

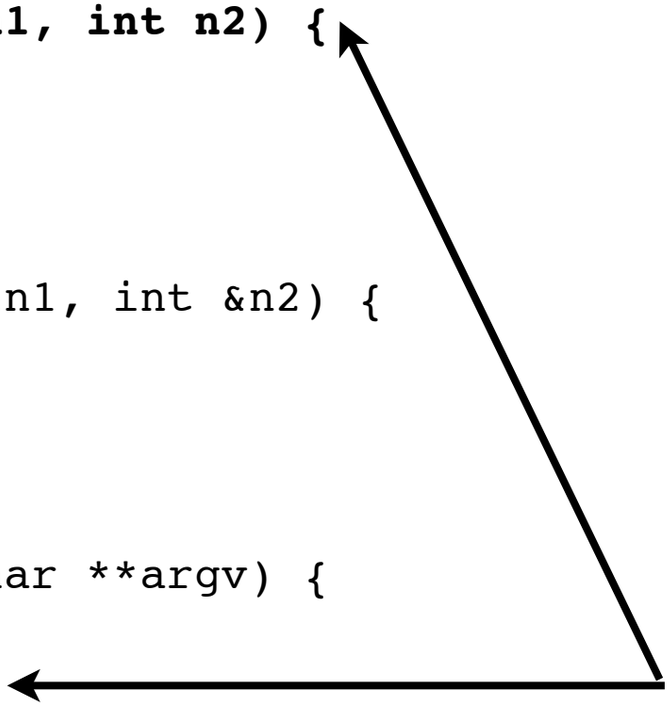
x	y
5	3

Call Stack:

main : 15
swap_wrong : 03

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```



We don't have a register waiting for a return value, but we're still calling the function.

Just like with the max function, we add swap_wrong to the call stack, create the registers for it's arguments (n1 and n2) and assign their values to x and y.

Registers for 'main':

x	y
5	3

Registers for 'swap_wrong':

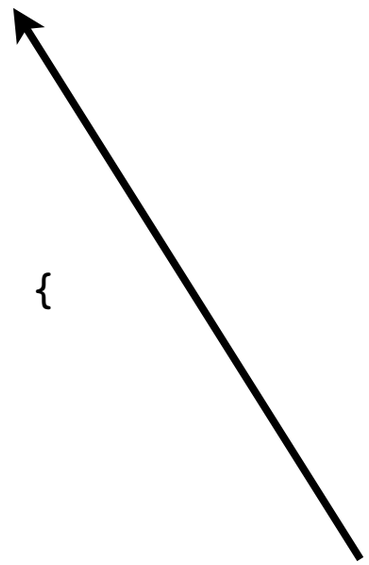
n1	n2
5	3

Call Stack:

main : 15
swap_wrong : 03

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << “x: ” << x << “, y: “ << y << endl;
17   swap_right(x, y);
18   cout << “x: ” << x << “, y: “ << y << endl;
19 }
```



Now execution continues from line 03.

Registers for ‘main’:

x	y
5	3

Call Stack:

main : 15
swap_wrong : 03

Registers for ‘swap_wrong’:

n1	n2
5	3

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << “x: ” << x << “, y: “ << y << endl;
17   swap_right(x, y);
18   cout << “x: ” << x << “, y: “ << y << endl;
19 }
```

Line 04 creates a new register for temp, and copies the data from n1 into that register.

Registers for ‘main’:

x	y
5	3

Call Stack:

main : 15
swap_wrong : 04

Registers for ‘swap_wrong’:

n1	n2	temp
5	3	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << “x: ” << x << “, y: “ << y << endl;
17   swap_right(x, y);
18   cout << “x: ” << x << “, y: “ << y << endl;
19 }
```

Line 05 copies the value from the n2 register into the n1 register.

Registers for ‘main’:

x	y
5	3

Call Stack:

main : 15
swap_wrong : 04

Registers for ‘swap_wrong’:

n1	n2	temp
3	3	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << “x: ” << x << “, y: “ << y << endl;
17   swap_right(x, y);
18   cout << “x: ” << x << “, y: “ << y << endl;
19 }
```

Line 06 copies the value from the temp register into the n2 register.

Registers for ‘main’:

x	y
5	3

Call Stack:

main : 15
swap_wrong : 04

Registers for ‘swap_wrong’:

n1	n2	temp
3	5	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << “x: ” << x << “, y: “ << y << endl;
17   swap_right(x, y);
18   cout << “x: ” << x << “, y: “ << y << endl;
19 }
```

The function finishes on line 07, which removes all the registers for swap_wrong and brings us back into the main method where we left off.

Registers for ‘main’:

x	y
5	3

Call Stack:

main : 15
swap_wrong : 04

Registers for ‘swap_wrong’:

n1	n2	temp
3	5	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

The function finishes on line 07, which removes all the registers for swap_wrong and brings us back into the main method where we left off.

Registers for 'main':

x	y
5	3

Call Stack:

main : 15

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Note that now we have a problem, we changed the values of the registers for the swap_wrong function, but nothing changed with the registers for our main function! The swap_wrong function doesn't work!

Registers for 'main':

x	y
5	3

Call Stack:

main : 15

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

When we print out x and y, it still prints out 5 and 3, because the x and y registers of the main function never got changed.

Luckily c++ has a solution to this problem (although some computer scientists may argue this is a bad thing).

Registers for 'main':

x	y
5	3

Call Stack:

main : 16

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

If you put a & before a parameter in a function, as in the swap_right function, the parameters to the function are passed by reference (instead of by value).

Registers for 'main':

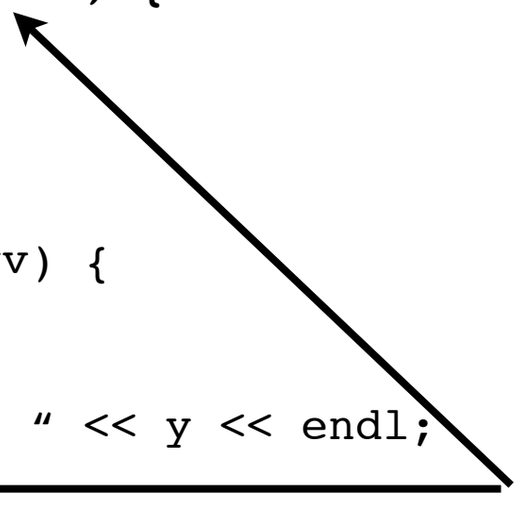
x	y
5	3

Call Stack:

main : 17

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```



This means, when the flow of our program goes into the swap_right function, the swap_right function is using **the same** registers as were passed into the function. n1 and x are the same, and n2 and y are the same.

Registers for 'main':

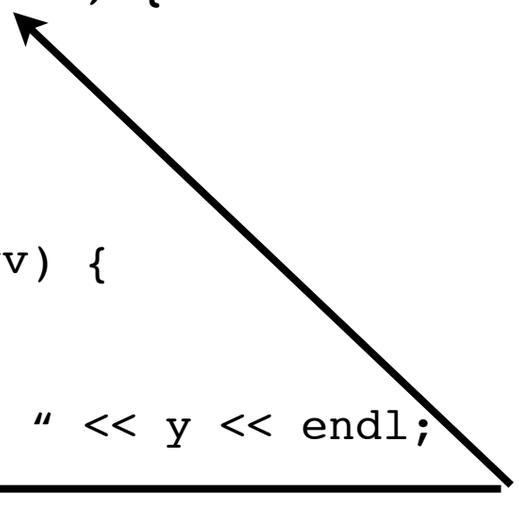
x	y
5	3

Call Stack:

main : 17

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```



This means, when the flow of our program goes into the swap_right function, the swap_right function is using **the same** registers as were passed into the function. n1 and x are the same, and n2 and y are the same.

Registers for 'main':

x	y
5	3

Registers for 'swap_right':

n1	n2
reference to x	reference to y

Call Stack:

main : 17
swap_right : 08

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Now the program keeps going from the beginning of the swap_right function.

Registers for 'main':

x	y
5	3

Registers for 'swap_right':

n1	n2
reference to x	reference to y

Call Stack:

main : 17
swap_right : 08

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

We create a new register called 'temp' and assign it to the value of n1 -- which is the value at the x register.

Registers for 'main':

x	y
5	3

Call Stack:

main : 17
swap_right : 09

Registers for 'swap_right':

n1	n2	temp
reference to x	reference to y	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Now when we assign n2 to n1, it is copying the value from the y register into the x register; because n2 is a reference to y and n1 is a reference to x.

Registers for 'main':

x	y
3	3

Call Stack:

main : 17
swap_right : 10

Registers for 'swap_right':

n1	n2	temp
reference to x	reference to y	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

And when we assign the value of temp to n2, we're actually setting the value at the y register, because n2 is a reference to y.

Registers for 'main':

x	y
3	5

Call Stack:

main : 17
swap_right : 11

Registers for 'swap_right':

n1	n2	temp
reference to x	reference to y	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

When the function finishes, the registers used by swap right are removed -- in this case the two references to x and y (n1 and n2, respectively) as well as temp.

Registers for 'main':

x	y
3	5

Call Stack:

main : 17
swap_right : 12

Registers for 'swap_right':

n1	n2	temp
reference to x	reference to y	5

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

When the function finishes, the registers used by swap right are removed -- in this case the two references to x and y (n1 and n2, respectively) as well as temp.

Registers for 'main':

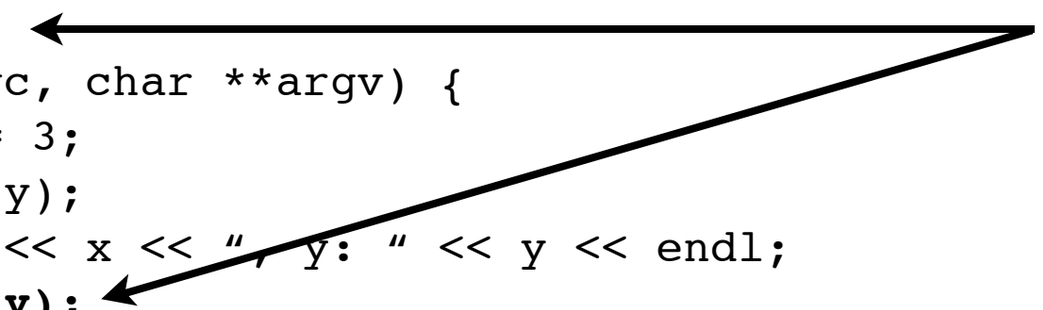
x	y
3	5

Call Stack:

main : 17
swap_right : 12

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```



Following this, we pop the function off the call stack and go back to where we left off.

Registers for 'main':

x	y
3	5

Call Stack:

main : 17

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Following this, we pop the function off the call stack and go back to where we left off.

Registers for 'main':

x	y
3	5

Call Stack:

main : 17

Call by Reference

```
01 #include <iostream>
02 using namespace std;
03 void swap_wrong(int n1, int n2) {
04   int temp = n1;
05   n1 = n2;
06   n2 = temp;
07 }
08 void swap_right(int &n1, int &n2) {
09   int temp = n1;
10   n1 = n2;
11   n2 = temp;
12 }
13 int main(int argc, char **argv) {
14   int x = 5, y = 3;
15   swap_wrong(x, y);
16   cout << "x: " << x << ", y: " << y << endl;
17   swap_right(x, y);
18   cout << "x: " << x << ", y: " << y << endl;
19 }
```

Now when we print out the values of x and y, they are actually swapped. This is because the parameters passed to swap_right were call by reference (because of the &s); unlike in swap_wrong, which is call by value (the default case).

Registers for 'main':

x	y
3	5

Call Stack:

main : 18