We are 183

L04: Week 3 - Wednesday

Carnival

“Welcome to Computing” Carnival!

WHEN: Thursday, January 21, 7:30-10PM

WHERE: BOB & BETTY BEYSTER BLDG

WHAT: FOOD, FUN, GAMES
MIX AND MINGLE WITH YOUR PEERS
WIN PRIZES

FOR STUDENTS IN ENGR 101, EECS 183 & EECS 280

HKN

Engineering Career Fair: January 26 – 27, 1-6 pm

- North Campus
- Hundreds of Companies
- Go talk to them
- Discover what they are looking for

Tutoring Available – Now!

- Free tutoring!
- **Appointments** -> Tutoring tab on course website
- Half-hour, one-on-one, up to once per week
- What tutoring **is**:
  - 30 minutes of help on any topic(s) of your choosing
- What tutoring is **not**:
  - 30 minute office hour session to troubleshoot your project code
- Location: Duderstadt Center, North campus

Last Time… on EECS 183

Casting, Imprecision

Compile and run-time Errors

Testing & Debugging

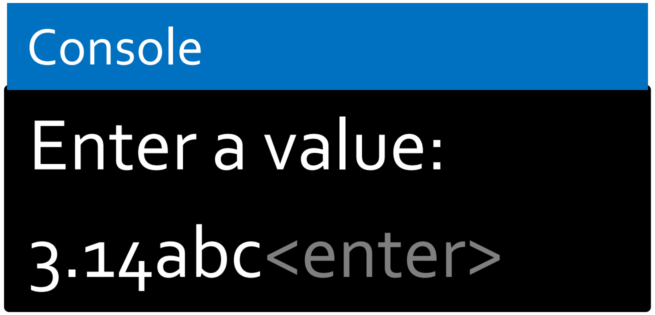
Pre-defined Functions

cin, cout

i>Clicker #1

What prints?

```
cout << "Enter a value: ";
int x = 5;
double z = 0;
char ch = ' ';

cin >> x >> z >> ch;
cout << x << "  "
     << z << "  "
     << ch << endl;
```



Console

```
Enter a value:
3.14abc<enter>
```

A) 5 0 3
B) 3 14 .
C) 3 0.14 a
D) None of the above

i>Clicker #1

What prints?

```
cout << "Enter a value: ";
int x = 5;
double z = 0;
char ch = ' ';

cin >> x >> z >> ch;
cout << x << "  "
     << z << "  "
     << ch << endl;
```

Console

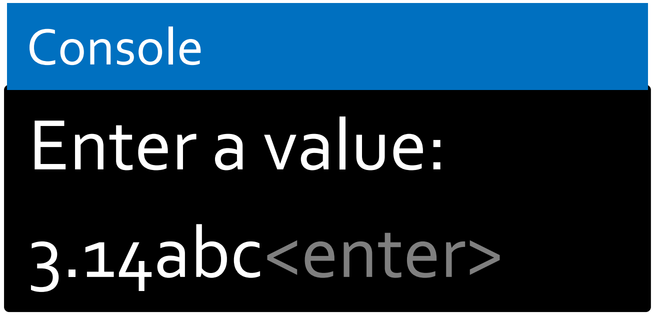
```
Enter a value:
3.14abc<enter>
```

```
A) 5  0  3
B) 3  14  .
C) 3  0.14  a
```
D) None of the above

i>Clicker #2

What prints?

```
cout << "Enter a value: ";
int x = 5;
double z = 0;
char ch = ' ';

cin >> x >> ch >> z;
cout << x << "  "
     << z << "  "
     << ch << endl;
```



```
Console
Enter a value:
3.14abc<enter>
```

A) 5 0 3

B) 3 14 .

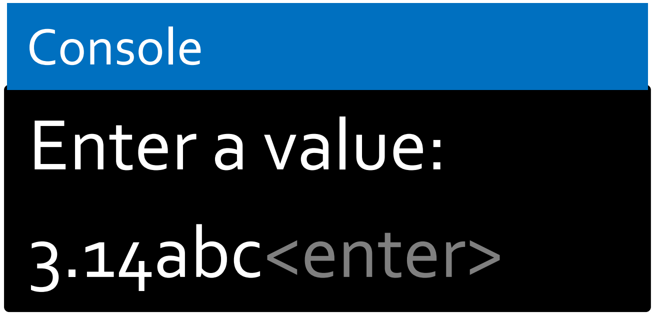
C) 3 0.14 a

D) None of the above

i>Clicker #2

What prints?

```
cout << "Enter a value: ";
int x = 5;
double z = 0;
char ch = ' ';

cin >> x >> ch >> z;
cout << x << "  "
     << z << "  "
     << ch << endl;
```



Console

```
Enter a value:
3.14abc<enter>
```

A) 5 0 3

B) 3 14 .

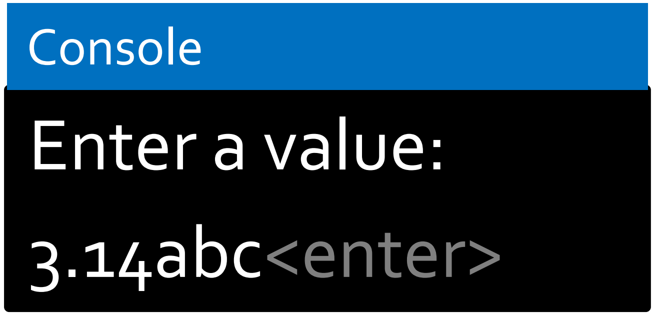
C) 3 0.14 a

D) None of the above

i>Clicker #3

What prints?

```
cout << "Enter a value: ";
int x = 5;
double z = 0;
char ch = ' ';

cin >> ch >> x >> z;
cout << x << "   "
     << z << "   "
     << ch << endl;
```



```
Enter a value:
3.14abc<enter>
```

A) 5   0   3

B) 0   0   3

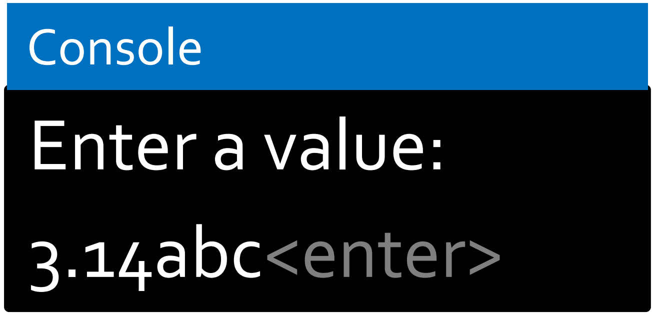
C) 3   0.14   a

D) A or B

i>Clicker #3

What prints?

```
cout << "Enter a value: ";
int x = 5;
double z = 0;
char ch = ' ';

cin >> ch >> x >> z;
cout << x << "  "
     << z << "  "
     << ch << endl;
```



Console

```
Enter a value:
3.14abc<enter>
```

A) 5 0 3

B) 0 0 3

C) 3 0.14 a

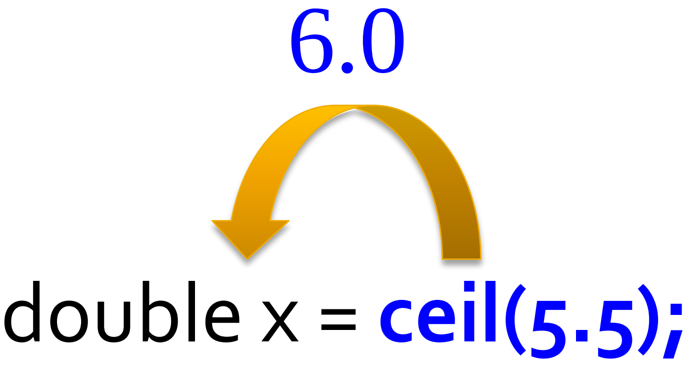
D) A or B

Pre-defined Functions

- A **function** is a list of statements that can be executed by referring to the function's name
- An input value to the function appears between ( )
  - Called arguments
- The function executes and returns a new value

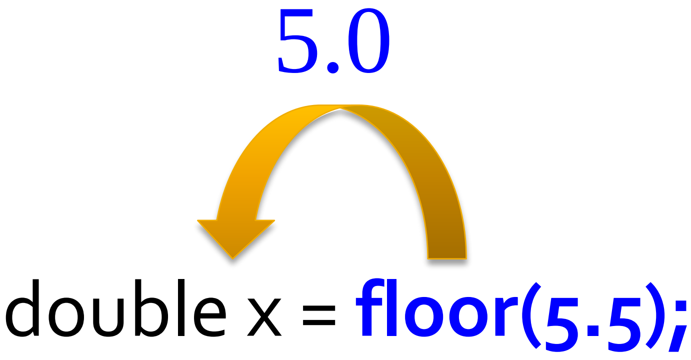
<cmath> Functions

- Functions **execute** and **return a new value**



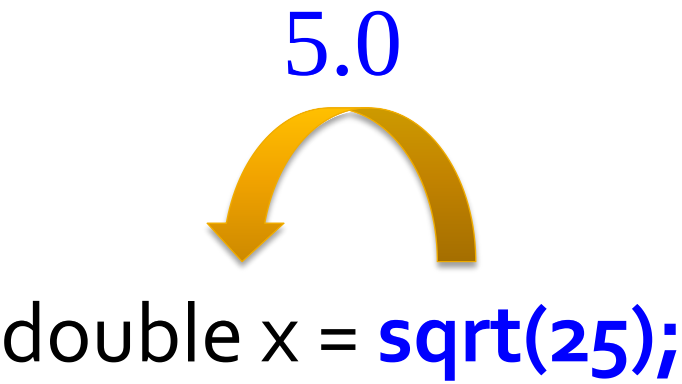
6.0

double x = **ceil(5.5);**



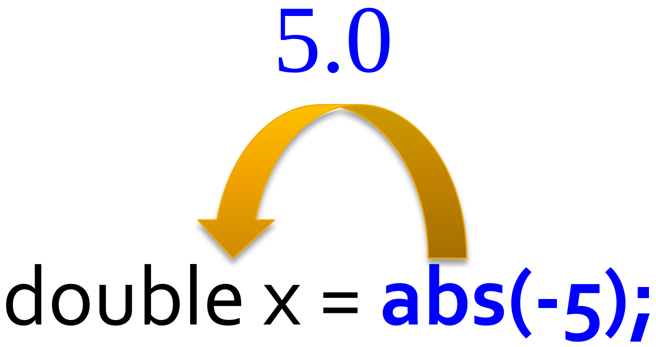
5.0

double x = **floor(5.5);**



5.0

double x = **sqrt(25);**



5.0

double x = **abs(-5);**

i>Clicker #4

```
int main() {
    double x = 5.0001;
    double y = 5.9999;
    cout << ceil(x) + 2 * floor(y);
}
```

This program prints:

A. 15
B. 16
C. 17
D. 18
E. None of the above

i>Clicker #4

```
int main() {
    double x = 5.0001;
    double y = 5.9999;
    cout << ceil(x) + 2 * floor(y);
}
```

This program prints:

A. 15

B. 16

C. 17

D. 18

E. None of the above

i>Clicker #5

```
int main() {
    double x = 6.0;
    double y = 5.0;
    cout << ceil(x) + 2 * floor(y);
}
```

This program prints:

A. 15
B. 16
C. 17
D. 18
E. None of the above

i>Clicker #5

```
int main() {
    double x = 6.0;
    double y = 5.0;
    cout << ceil(x) + 2 * floor(y);
}
```

This program prints:

A. 15
B. 16
C. 17
D. 18
E. None of the above

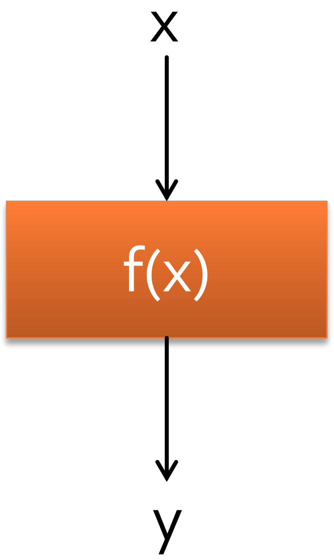
Today

User-Defined Functions
Unit Testing
Requires, Modifies, Effects (RMEs)
Global and Local Variables
Scope

What's a function?

- In mathematics:
  - For every input, there is exactly one output

$$\overset{\text{Output}}{y} = f(\overset{\text{Input}}{x})$$

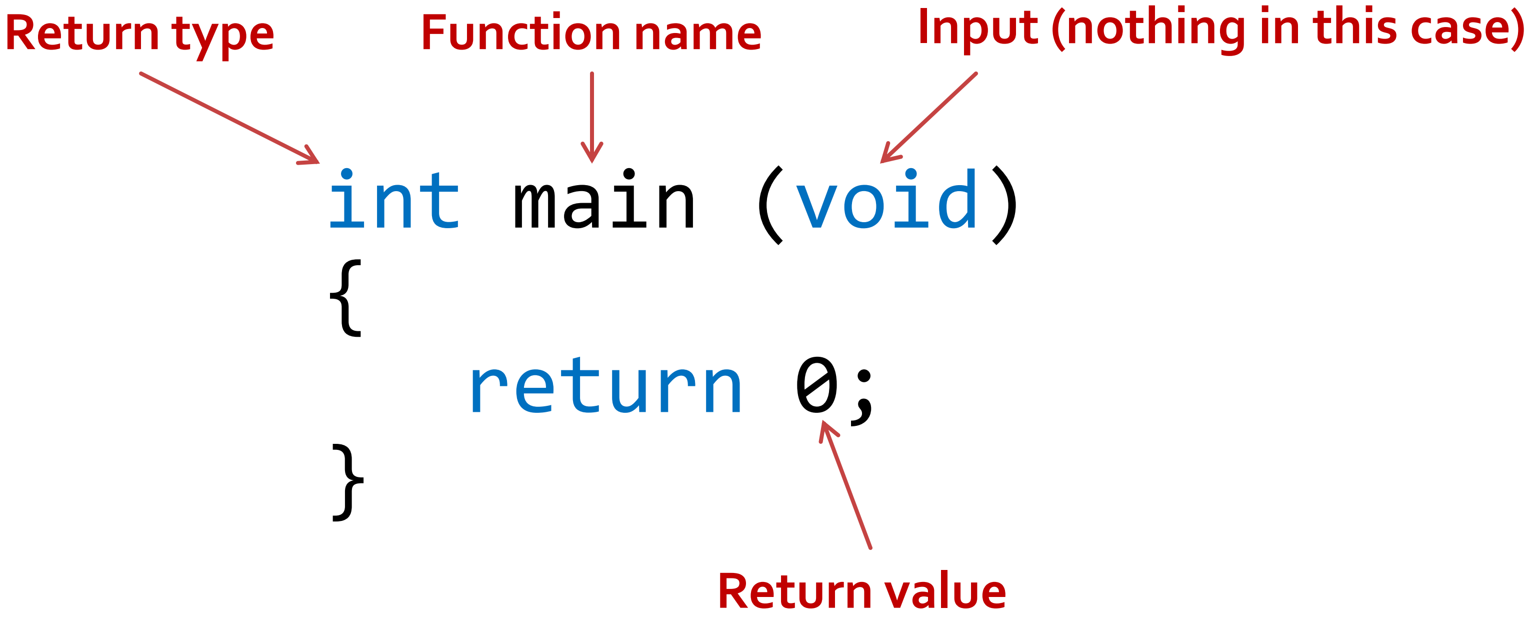


What’s a function?

- In Programming:
  - A section of a program
  - that can act on data and
  - returns a value

Functions in Programming

- We have already seen the main function



Return type

Function name

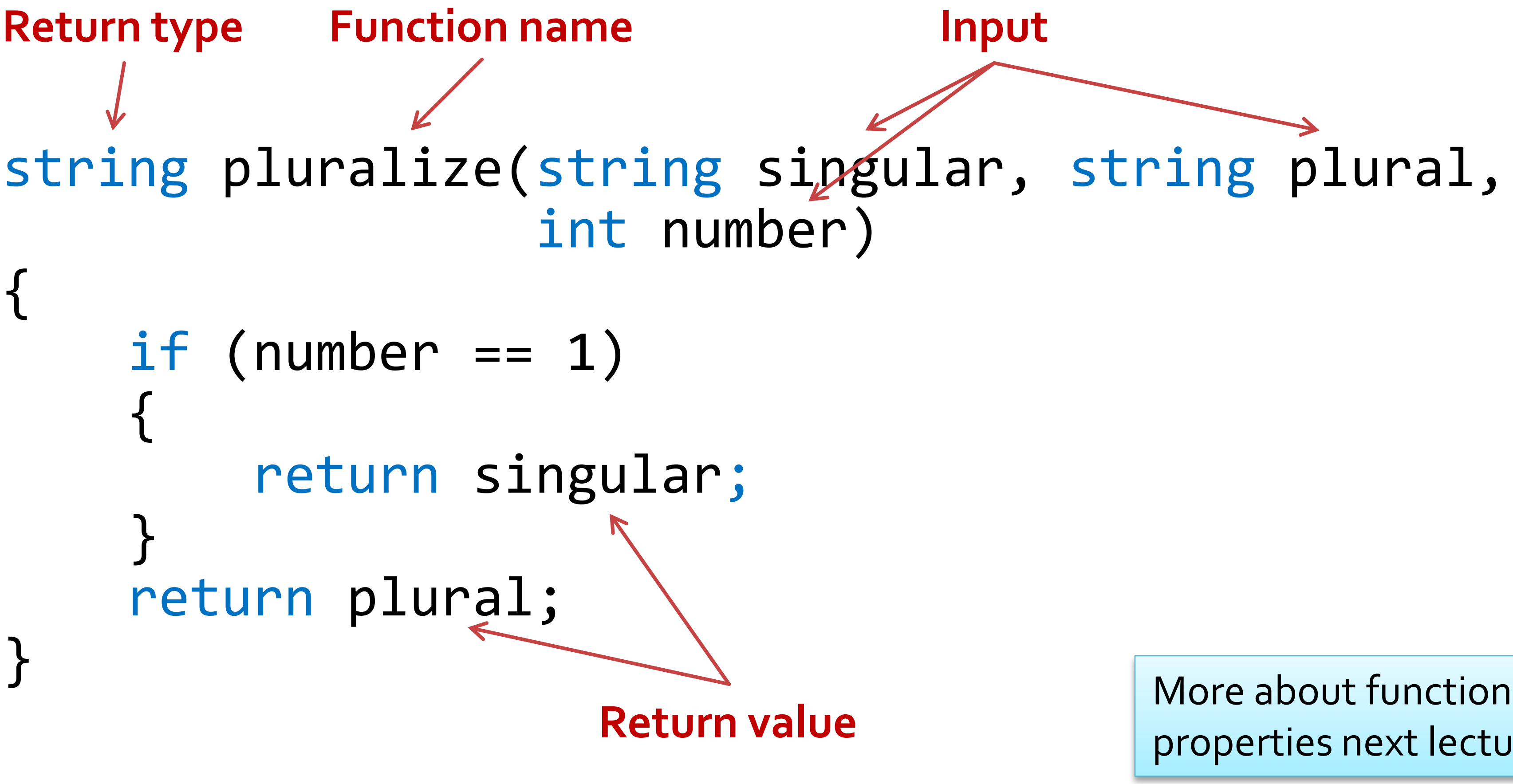
Input (nothing in this case)

```
int main (void)
{
    return 0;
}
```

Return value

Functions in Programming

- Example: `pluralize` function



Return type

Function name

Input

```
string pluralize(string singular, string plural,
                 int number)
{
    if (number == 1)
    {
        return singular;
    }
    return plural;
}
```

Return value

More about function properties next lecture

Active Demo: `pluralize` Function

- We are going to perform a live, choreographed demonstration

- Need two student volunteers:
    - `main()`
    - `pluralize()`

User Defined Functions

- Every function you would ever want Doesn't exist
- At some point you need to create your own

Why Functions???

- Want, Code that is Easier to
  - Read
  - Maintain
  - Test
  - Develop
- No code duplication

- Length of program, not as important

Why User-Defined Functions

- Reusability
- Readability
- Reduce bugs
- Division of labor
- Naturally fits with pseudo code

How do you know it's a fn?

- Look for the ()
  immediately following identifier

  - sqrt (7)

How do you know it’s a fn?

- Look for the ()
  immediately following identifier

  - sqrt (7) fn
  - cout << sqrt(4)

How do you know it's a fn?

- Look for the ()
  immediately following identifier

  - sqrt (7)   fn
  - cout << sqrt(4)   fn
  - cos(0.4)

How do you know it’s a fn?

- Look for the ()
  immediately following identifier
  - sqrt (7) fn
  - cout << sqrt(4) fn
  - cos(0.4) fn
  - sin (pi/2)

How do you know it’s a fn?

- Look for the ()
  immediately following identifier

sqrt (7)	fn
cout << sqrt(4)	fn
cos(0.4)	fn
sin (pi/2)	fn
tan * (3)	

How do you know it’s a fn?

- Look for the ()
  immediately following identifier

sqrt (7)	fn
cout << sqrt(4)	fn
cos(0.4)	fn
sin (pi/2)	fn
tan * (3)	NOT fn

Intermission

Two-minute break

Example: square

- We are going to create and use our own function
- Function will calculate the square of an integer
- No operator exists for square
    - We can create a function!

Example: square

- What steps will we take to create our function?
  - **Define** the problem
  - Create an **algorithm** to solve the problem
  - Create a **test suite** to test our source code
  - Translate our algorithm to **source code**
  - **Test** our function

Problem Definition: square

- Problem definition

Problem Definition: square

- Problem definition
  - Provide square of a number (integer)
    - Function name, requirements, modifications, effects
  - Need to accomplish: the number to square
    - Function input
  - What do we get: The square of the number
    - Function return value and type

Problem Definition: square

- Problem definition
  - Provide square of a number (integer)
    - Function **name**, requirements, modifications, effects
  - Need to accomplish: the number to square
    - Function **input**
  - What do we get: The square of the number
    - Function **return** type

Algorithm: square

- Algorithm
  - Multiply the number by itself

$$f(x) = x^2$$

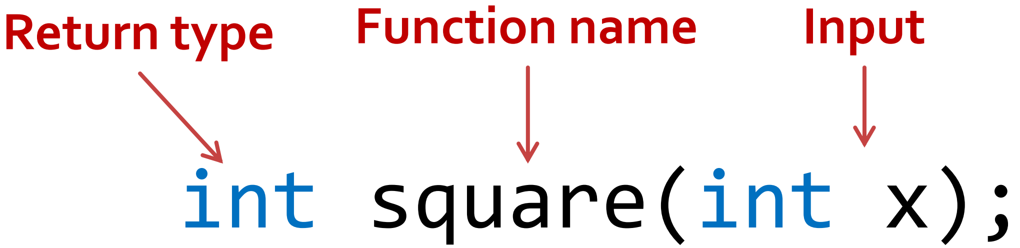
$$f(x) = x * x$$

Example: square

- We now have enough information to write the interface to our function
  - i.e., **function declaration**

Function Declaration: square

- square function declaration



Return type

Function name

Input

```
int square(int x);
```

Function Declaration

- Tells compiler
  - function **name**
  - type of data function produces (**return** type)
  - types of parameters (**inputs**)
- Also called prototype
- Contains NO code

Testing: square

- Testing – create a test suite
- Test suite: a battery of tests, the purpose of which is the verification of the correct functionality of a piece of code
  - In this case a function
- How to test? – **Call the function**

Call to: int square(int x)

-
  ```
  // Output return value
  cout << square(10);
  ```

-
  ```
  // Store return value
  int val = square(10);
  ```

-
  ```
  // Use return value
  int val = 5 * square(10);
  ```

If you don't OSU it – Doesn't get you much

```
// not Outputted
// not Stored
// not Used
square(10);

// but legal
```

Testing: square

-
  ```
  // Negative input
  cout << square(-10);
  ```

-
  ```
  // positive input
  cout << square(7);
  ```

-
  ```
  // zero input
  cout << square(0);
  ```

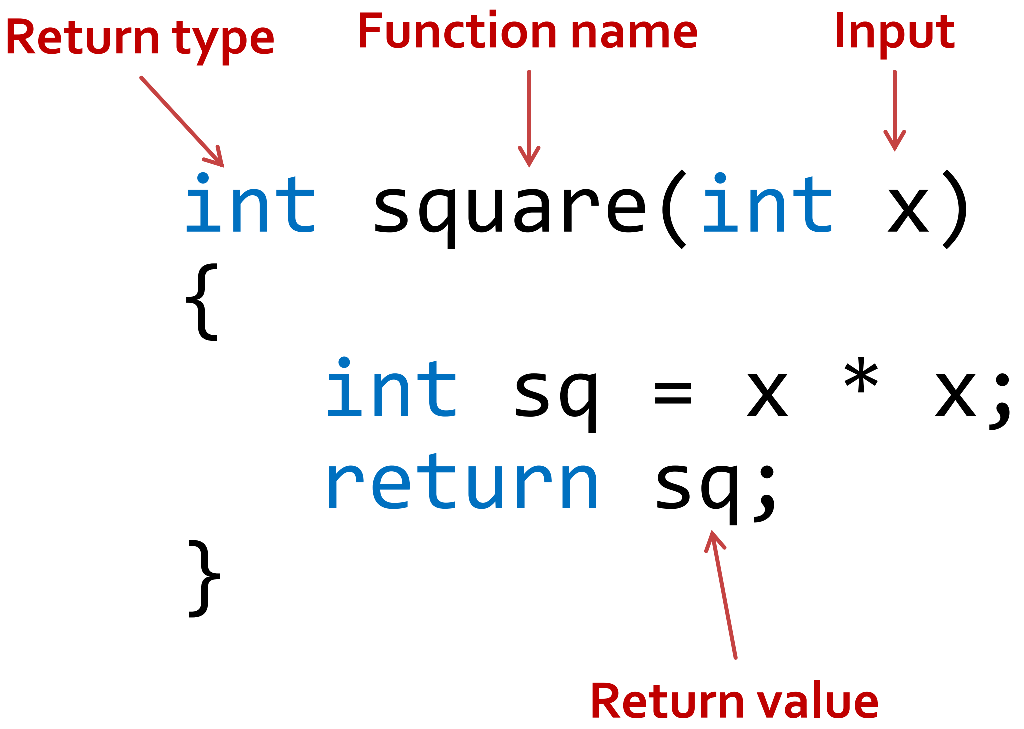
-
  ```
  // large input
  cout << square(10000);
  ```

More tests possible

Know the answer ahead of time!

Source Code: square

- Function **Definition**: square function



Return type

Function name

Input

```
int square(int x)
{
    int sq = x * x;
    return sq;
}
```

Return value

Intermission

Two-minute break

i>Clicker #6

What does the following print?

```
cout << sqrt(sqrt(16));
```

A) 1
B) 2
C) 4
D) None of the above

i>Clicker #6

What does the following print?

```
cout << sqrt(sqrt(16));
```

A) 1

B) 2

C) 4

D) None of the above

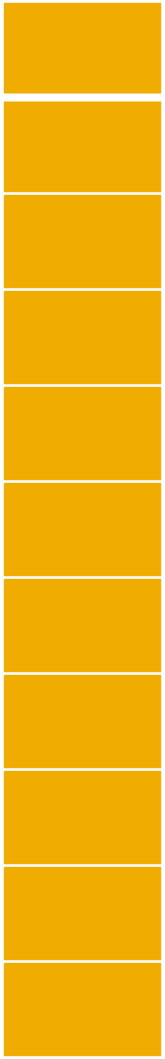
Pass by Value: scope

- We saw scope earlier in pluralize demo
- Variables only visible to function in which declared
  - Local variables
- Variables declared outside of any function?
  - Global variables
  - In this course, <u>must</u> be declared **const**

Pass-By-Value: scope

Execution

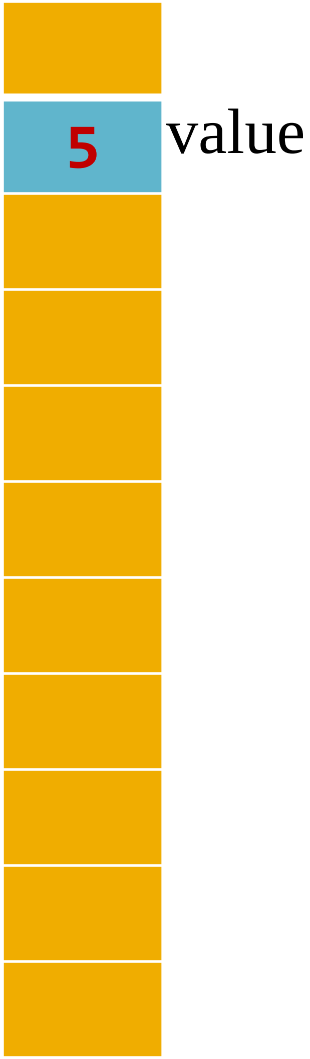
```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```



Pass-By-Value: scope

Execution

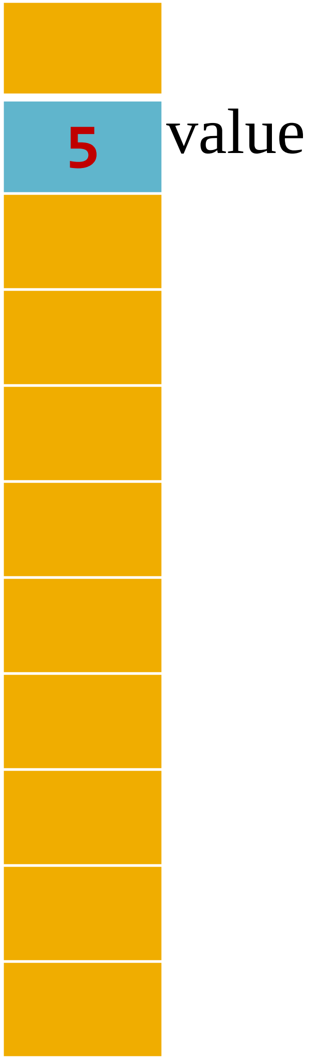
```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```



Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```

Execution

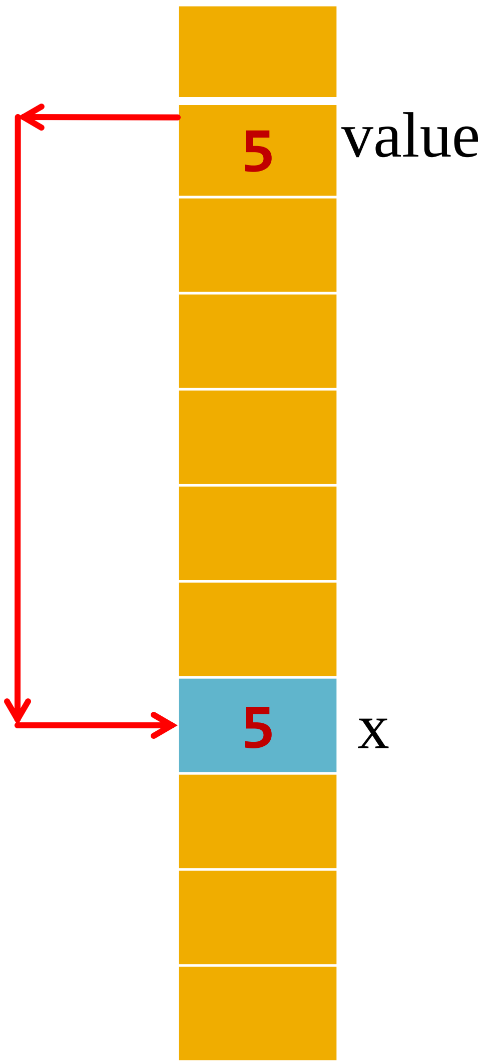


Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}
```

Execution

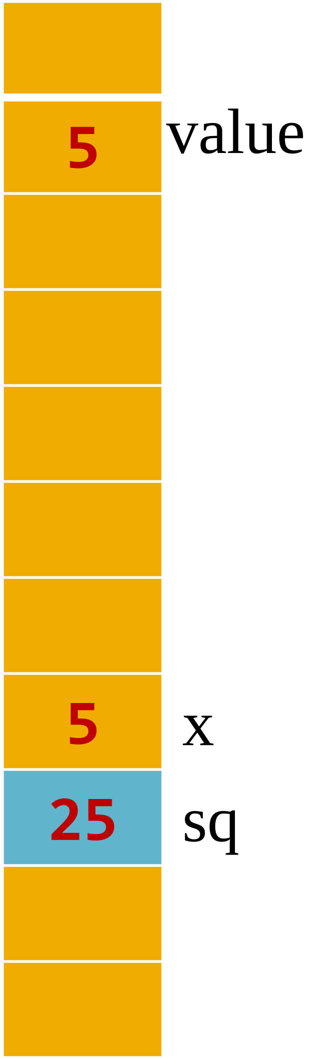
```
int square(int x)
{
    int sq = x * x;
    return sq;
}
```



Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```

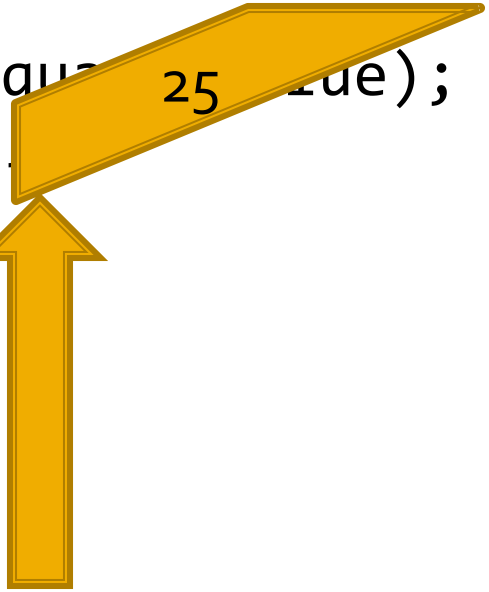
Execution

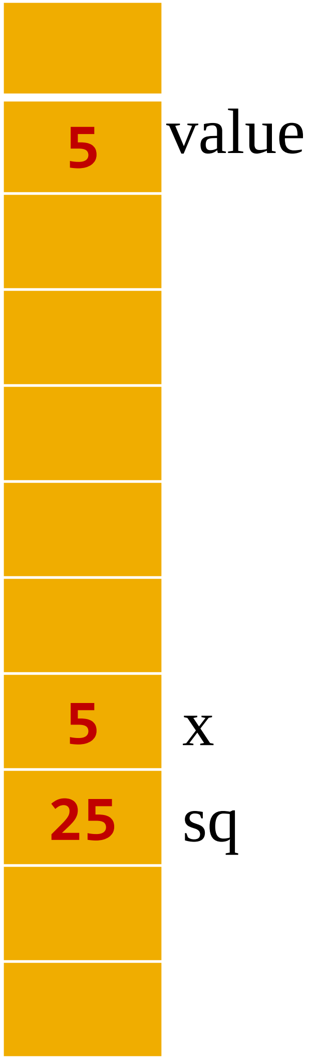


Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = squ        ue);
    cout << result
    return 0;
}

int square(int x)
{
    int sq = x * x
    return sq;
}
```

Execution

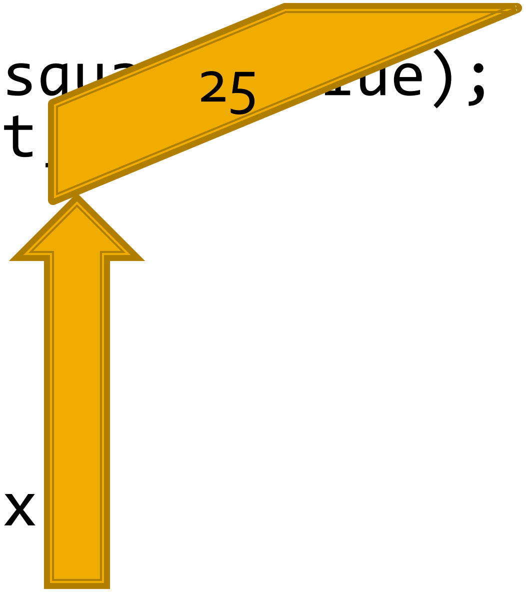


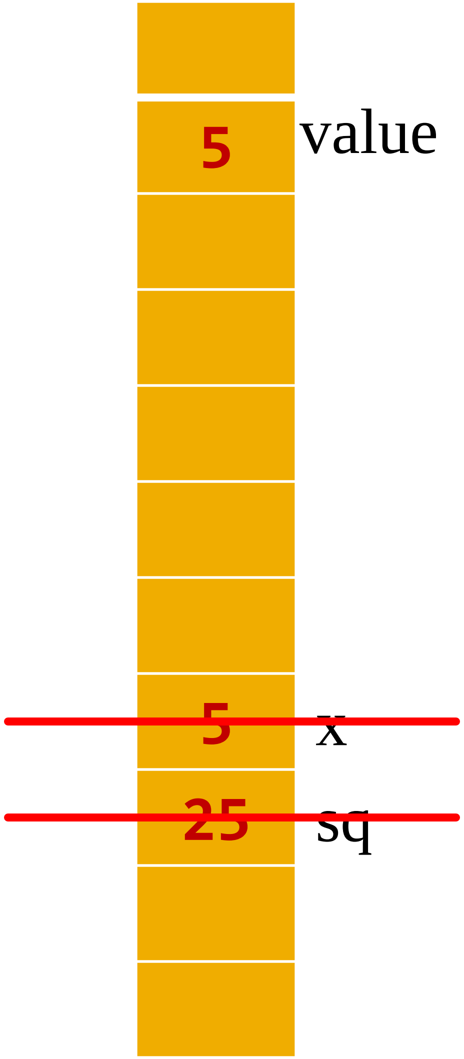


Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = squ        ue);
    cout << result
    return 0;
}

int square(int x)
{
    int sq = x * x
    return sq;
}
```

Execution

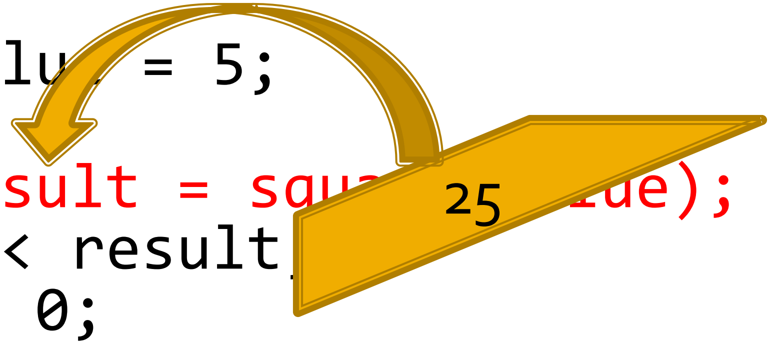


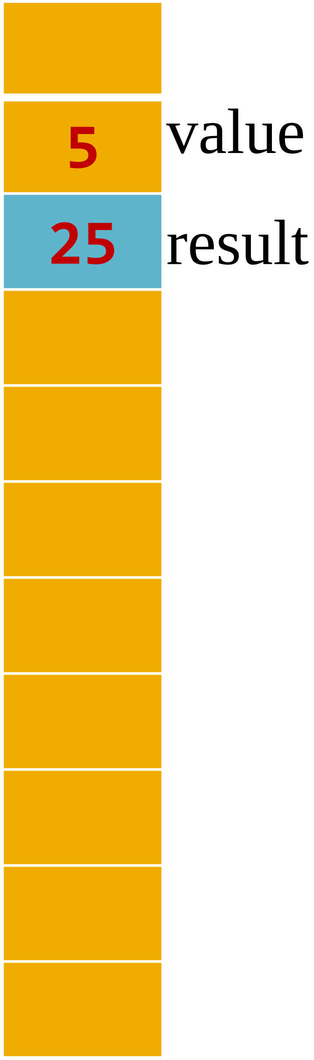


Pass-By-Value: scope

Execution

```
int main()
{
    int valu  = 5;

    int result = squ      ue);
    cout << result
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```

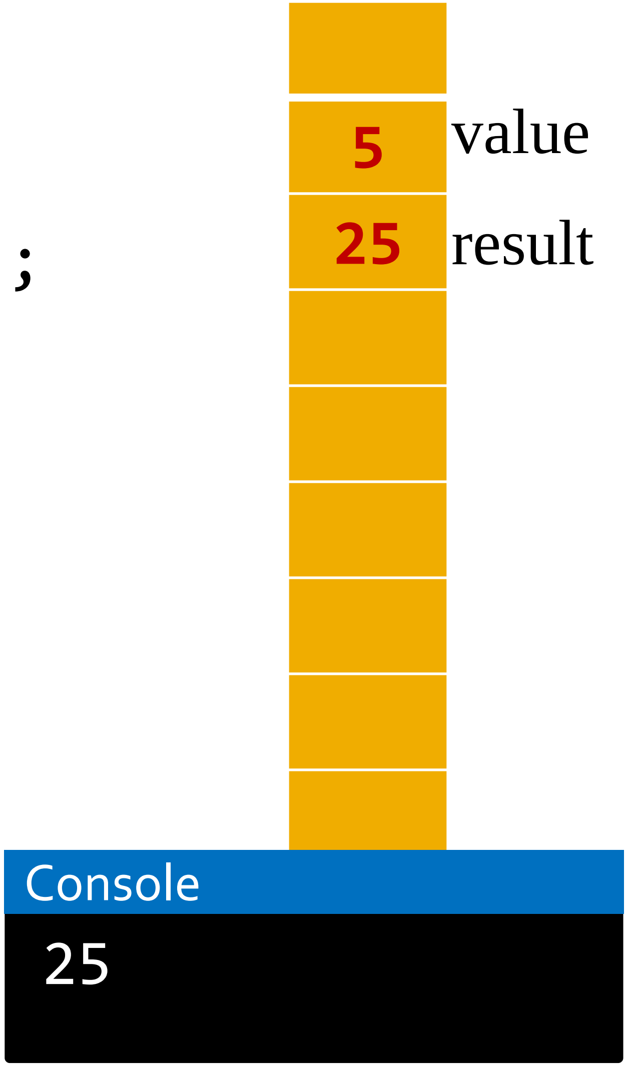




Pass-By-Value: scope

Execution

```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```

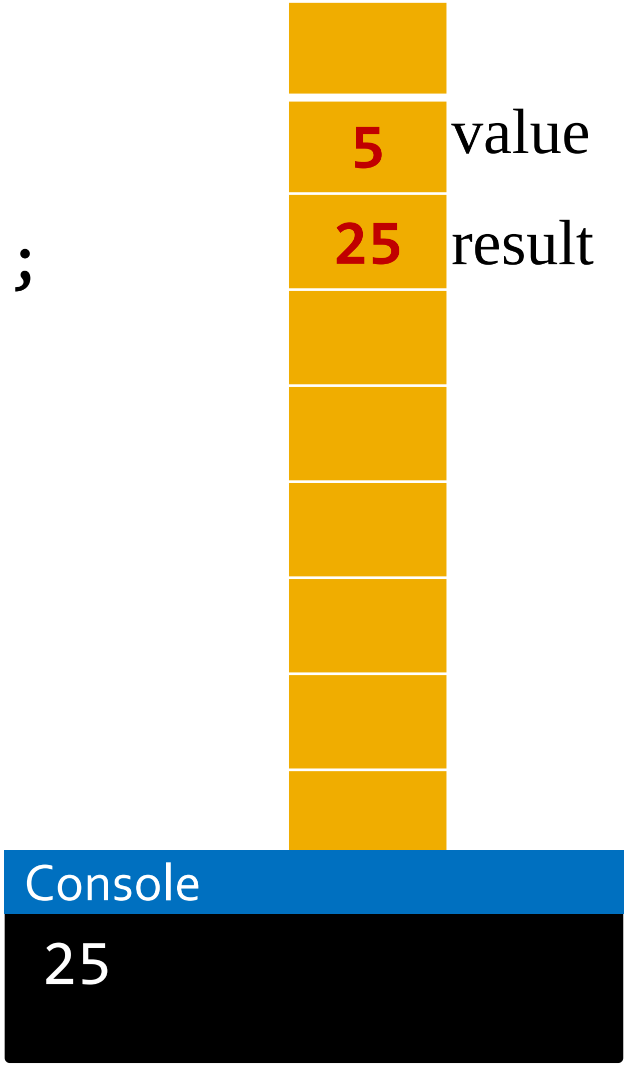


```
25
```

Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```

Execution



5 value

25 result

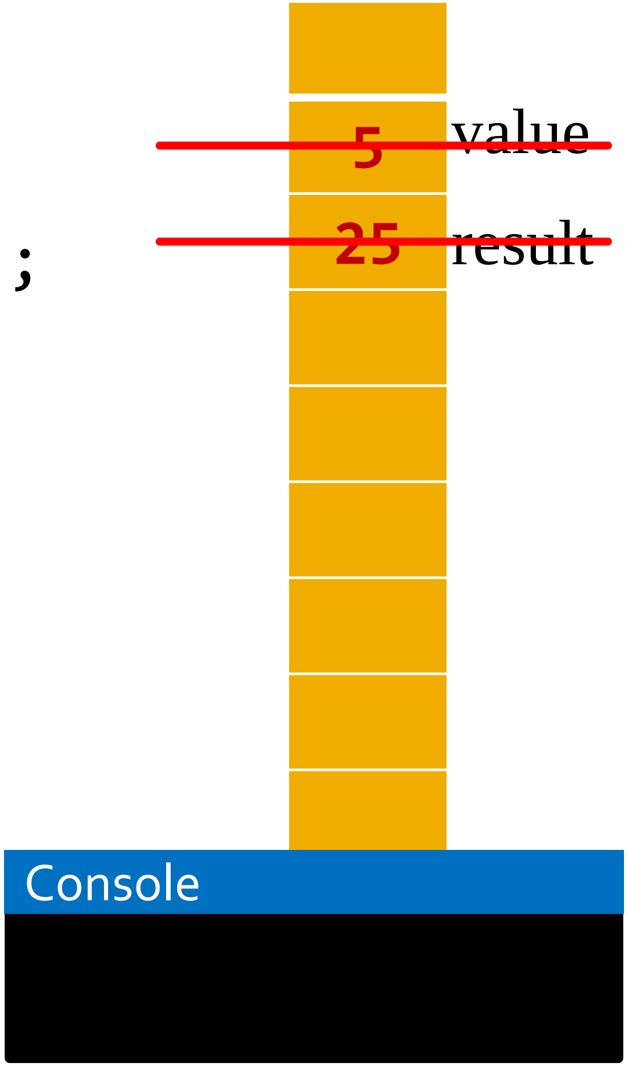
Console

```
25
```

Pass-By-Value: scope

```
int main()
{
    int value = 5;

    int result = square(value);
    cout << result;
    return 0;
}

int square(int x)
{
    int sq = x * x;
    return sq;
}
```

Execution



Variable Scope (visibility)

- Starts at declaration point

- Ends at the closing bracket of the enclosing block (e.g., end of function in the above example)

- Once the execution leaves the scope of a variable, the variable gets de-allocated (destroyed)

RME: what a function does, not how

- **Requires** – What inputs do the arguments take? Can they be any value, or are there additional constraints (for example, must be positive)?
- **Modifies** – Are the inputs going to be changed by the function? How are they going to be changed?
- **Effects** – What does the function do? What value is returned? Does it print to `cout`?

Exercise #1

Which of the following is a valid function prototype/declaration?

```
A. float some Function();
B. void nothing;
C. int (int thing);
D. void something();
```

Exercise #1

Which of the following is a valid function prototype/declaration?

A. `float some Function();`

B. `void nothing;`

C. `int (int thing);`

D. `void something();`

Exercise #2

Given the prototype : `void foo(int x);` which of the following calls are valid.

```
A. cout << foo(42);
B. int y = foo(15);
C. foo(-5);
D. int y = 5 + foo(6);
```

Exercise #2

Given the prototype : `void foo(int x);` which of the following calls are valid.

```
A. cout << foo(42);
B. int y = foo(15);
C. foo(-5);
D. int y = 5 + foo(6);
```

What does this print?

```cpp
#include <iostream>
using namespace std;

int main(void) {
    int x = 4;
    cout << x;
    {
        cout << x;
        int x = 3;
        cout << x;
    }
    cout << x;

    return 0;
}
```

A) 4333

B) 4334

C) 4444

D) 4434

E) Error

What does this print?

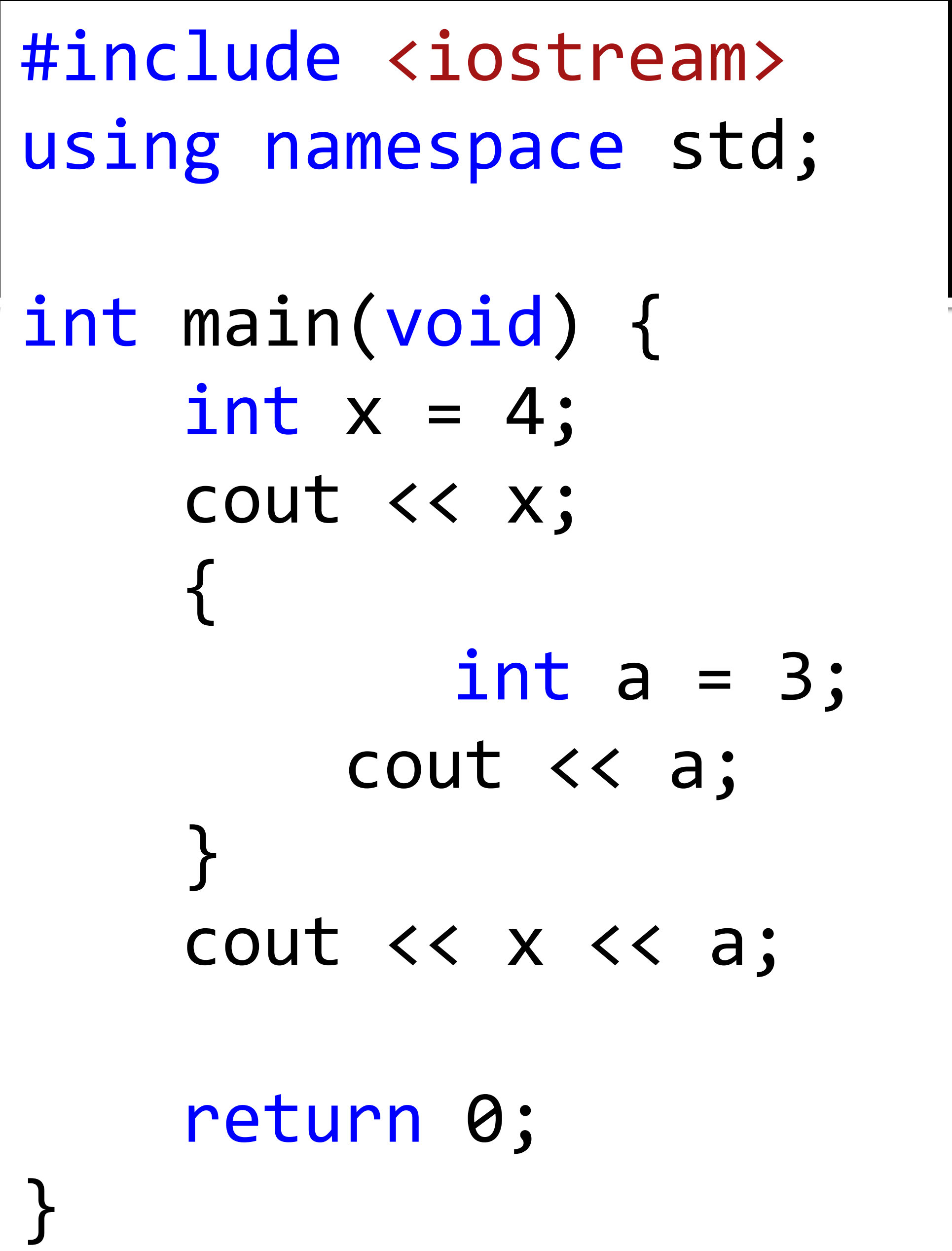
```
#include <iostream>
using namespace std;

int main(void) {
    int x = 4;
        cout << x;
    {
        cout << x;
        int x = 3;
        cout << x;
    }
    cout << x;

    return 0;
}
```

Scope of the inner x

Scope of the outer x

A) 4333
B) 4334
C) 4444
D) 4434
E) Error

What does this print?



```
#include <iostream>
using namespace std;

int main(void) {
    int x = 4;
    cout << x;
    {
           int a = 3;
        cout << a;
    }
    cout << x << a;

    return 0;
}
```

A) 4343
B) 4333
C) 3333
D) Error

What does this print?

```
#include <iostream>
using namespace std;

int main(void) {
    int x = 4;
    cout << x;
    {
           int a = 3;
        cout << a;
    }
    cout << x << a;

    return 0;
}
```

Scope of a

A) 4343
B) 4333
C) 3333
D) Error

a is not visible here. It is de-allocated (destroyed) after the execution leaves its scope

Homeward Bound

http://www.youtube.com/watch?v=UXEvZ8B04bE&feature=youtu.be