APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 10
February 6, 2012

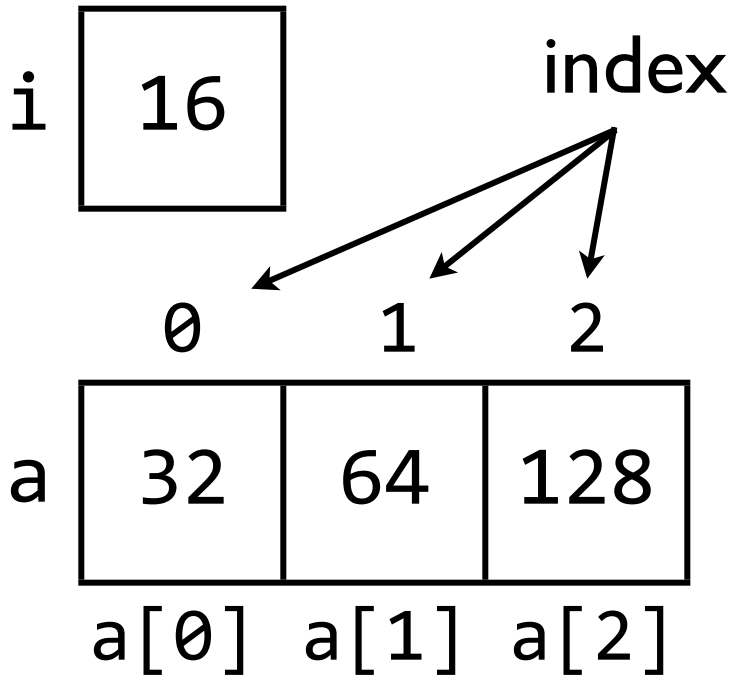
Today

- More Arrays
- Functions

Array

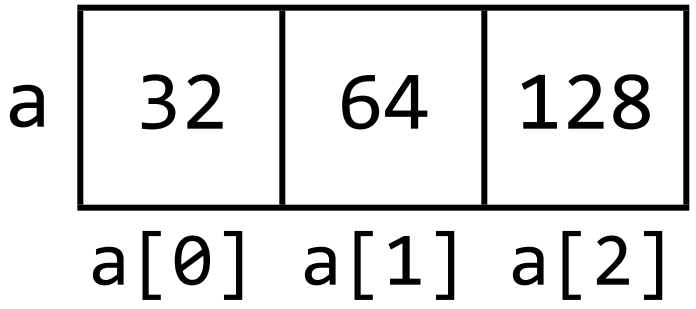
- A connected group of variables

```
int i;
i = 16;
```




```
int a[3];
a[0] = 32;
a[1] = 64;
a[2] = 128;
```

```
int a[3];
a[0] = 32;
a[1] = 64;
a[2] = 128;
```



What's the type of a? array of `int`

What's the type of `a[0]`? `int`

What's the type of `a[1]`? `int` } lvalues

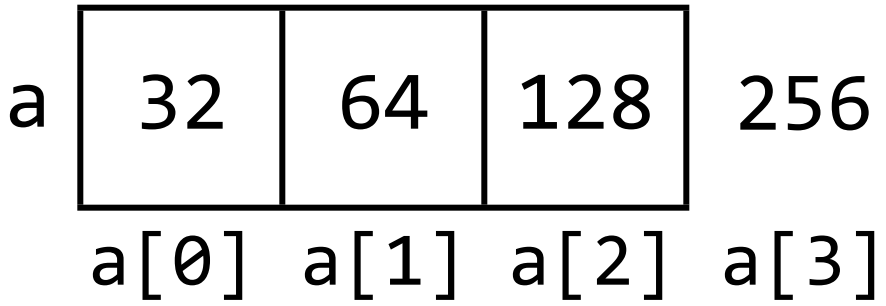
What's the type of `a[2]`? `int`

```
int x = 9;
a[0] = x + 2;
int sum = a[0] + a[1];
a[0]++;
a = 14;
```

Error

```
if (a[0] > MAX)
{
    ...
}
```

```
int a[3];
a[0] = 32;
a[1] = 64;
a[2] = 128;
a[3] = 256;
```



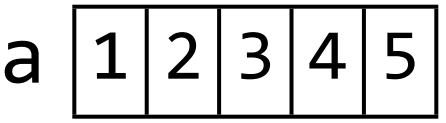
Initializing Arrays

```
int i;
i = 5;
```

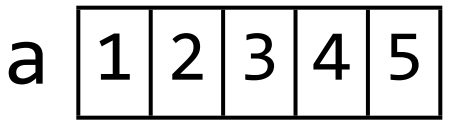
```
int i = 5;
```

```
int a[5];
a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;
a[4] = 5;
```

```
int a[5] = {1, 2, 3, 4, 5};
```




```
int a[]  = {1, 2, 3, 4, 5};
```



Iterating Arrays

```
#define N 10
...
int a[N];

for (int i = 0; i < N; i++)
{
    a[i] = i;
}

for (int i = 0; i < N; i++)
{
    printf("%d\n", a[i]);
}
```

Copying Arrays

- This will not work:

```
int a[4] = {1, 2, 3, 4};
int b[4];

b = a;
```

Error

```
int i = 8;
int j;

j = i;
```

Copying Arrays

```
int a[4] = {1, 2, 3, 4};
int b[4];

for (int i = 0; i < 4; i++)
{
    b[i] = a[i];
}
```

Other Types

```
bool b[3];
b[0] = true;
b[1] = false;
b[2] = b[0] && b[1];
```

```
char initials[3];
initials[0] = 'J';
initials[1] = 'A';
initials[2] = 'D';
```

```
double weight[3];
weight[0] = 3.4;
weight[1] = 2;
weight[2] = 3.1 + weight[0];
```

1-based indexing

- Arrays are `0`-based
- If you need 1-based indexing, just ignore `a[0]`

First place: 22.4 seconds	`times[0]`	`times[1]`
Second place: 24.9 seconds	`times[1]`	`times[2]`
Third place: 27.8 seconds	`times[2]`	`times[3]`

```
double times[3] = {22.4, 24.9, 27.8};

// Ignoring times[0] to provide 1-based indexing

double times[4] = {0.0, 22.4, 24.9, 27.8};
```

Multi-Dimensional Arrays

1-dimensional

0	1	3	5	7

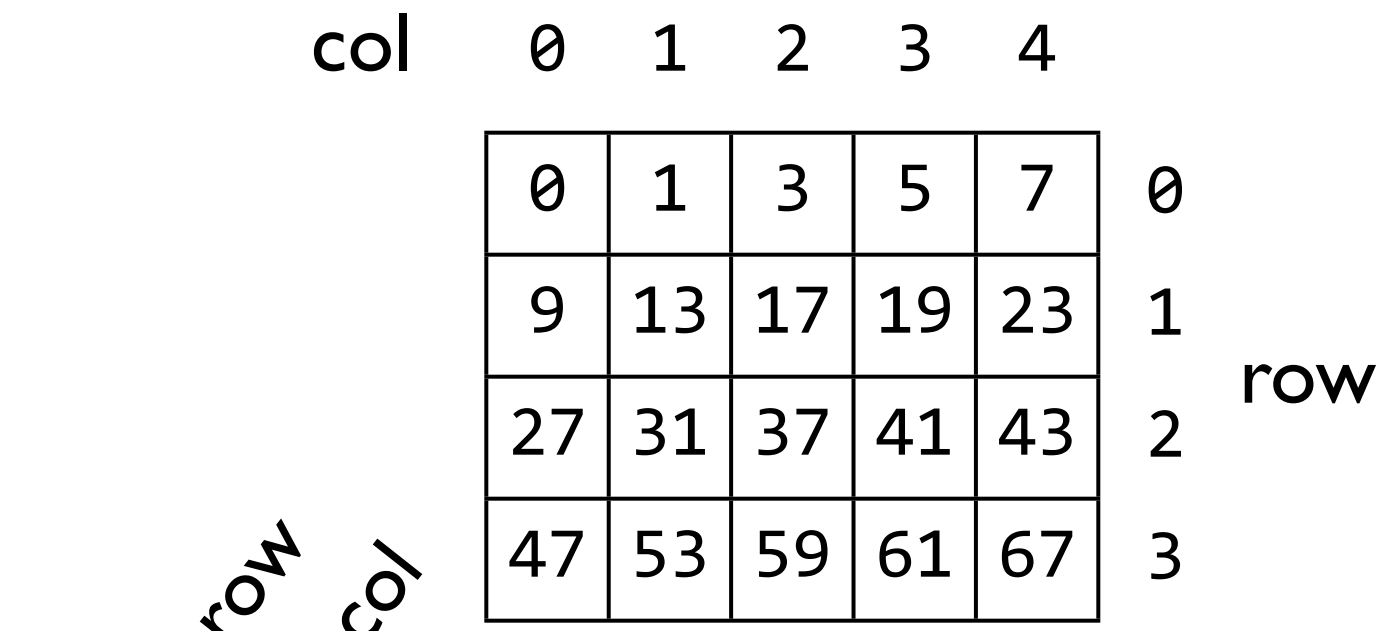
2-dimensional

0	1	3	5	7
9	13	17	19	23
27	31	37	41	43
47	53	59	61	67

```
int a[5];
```

```
int a[4][5];
```

row col




```
int a[4][5];
...
// Assume it gets filled in here
...
int i = a[2][0];
int j = a[1];
int k = a[3,2];
```

27

Wrong

Wrong

```
int m[3][4] = { {2, 4, 6, 8},
                {1, 3, 5, 7},
                {9, 8, 7, 6} };

int n[3][4] = { {2, 4, 6, 8},
                {1, 3, 5, 7} };

int n[3][4] = { {2, 4, 6, 8},
                {1, 3, 5, 7},
                {0, 0, 0, 0} };


int o[3][4] = { {2, 4, 6},
                {1, 3, 5},
                {9, 8, 7} };
int o[3][4] = { {2, 4, 6, 0},
                {1, 3, 5, 0},
                {9, 8, 7, 0} };
```

Multi-Dimensional Iteration

```
int m[N][N];
for (int row = 0; row < N; row++)
{
    for (int col = 0; col < N; col++)
    {
        if (row == col)
        {
            m[row][col] = 1;
        }
        else
        {
            m[row][col] = 0;
        }
    }
}
```

Multi-Dimensional Iteration

```
for (int row = 0; row < N; row++)
{
    for (int col = 0; col < N; col++)
    {
        printf("%d ", m[row][col]);
    }
    printf("\n");
}
```

```
****
***
**
*
```

height: 4

height rows

height - i "*" in row i

(0-based index)

```
for (int i = 0; i < height; i++)
{
  for (int j = 0; j < height - i; j++)
  {
      printf("*");
  }

  printf("\n");
}
```

```
****
***
**
*
```

height: 4

height rows

height - i "*" in row i

(0-based index)

```
char triangle[height][height];

for (int i = 0; i < height; i++)
{
  for (int j = 0; j < height - i; j++)
  {
      printf("*");
  }

  printf("\n");
}
```

```
****
***
**
*
```

height: 4

height rows

height - i "*" in row i

(0-based index)

```
char triangle[height][height];

for (int i = 0; i < height; i++)
{
  for (int j = 0; j < height - i; j++)
  {
      triangle[i][j] = "*";
  }

  printf("\n");
}
```

```
****
***
**
*
```

height: 4

height rows

height - i "*" in row i

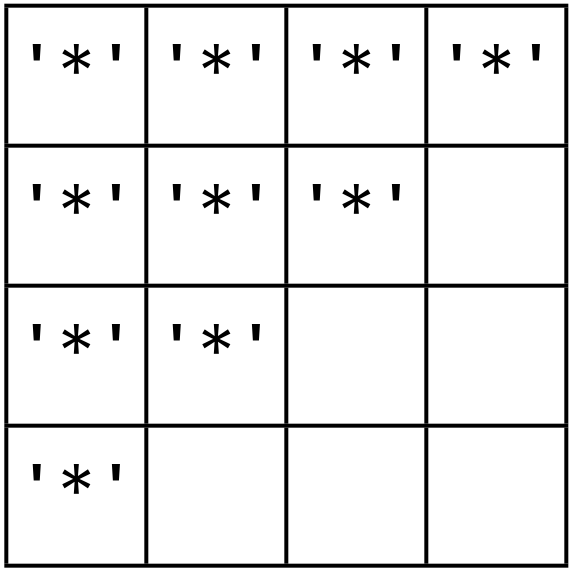
(0-based index)

```
char triangle[height][height];

for (int i = 0; i < height; i++)
{
  for (int j = 0; j < height - i; j++)
  {
      triangle[i][j] = '*';
  }

  printf("\n");
}
```

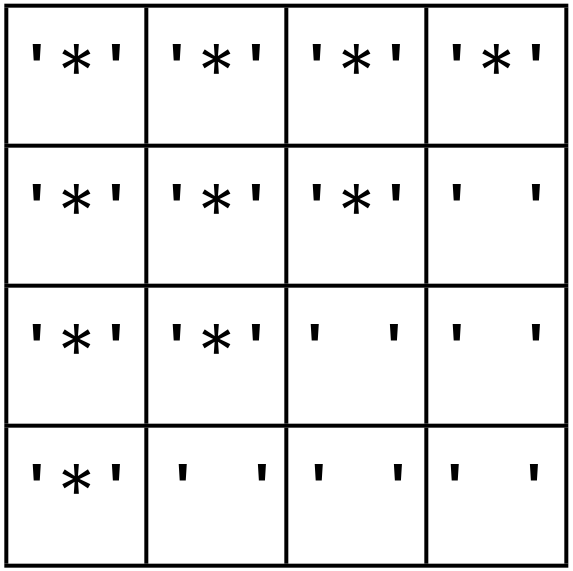
triangle



triangle

'*'	'*'	'*'	'*'
'*'	'*'	'*'	?
'*'	'*'	?	?
'*'	?	?	?

triangle



```
****
***
**
*
```

height: 4

height rows

height - i "*" in row i

and i " " in row i

(0-based index)

```
char triangle[height][height];

for (int i = 0; i < height; i++)
{
  for (int j = 0; j < height - i; j++)
  {
      triangle[i][j] = '*';
  }

  printf("\n");
}
```

```
****
***
**
*
```
height: 4

height rows (0-based index)
height - i "*" in row i
and i " " in row i

```
char triangle[height][height];
for (int i = 0; i < height; i++)
{
  int j;
  for (j = 0; j < height - i; j++)
  {
      triangle[i][j] = '*';
  }
  for ( ; j < height; j++)
  {
      triangle[i][j] = ' ';
  }
  printf("\n");
}
```

```
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < height; j++)
    {
        printf("%c", triangle[i][j]);
    }

    printf("\n");
}
```

Functions

Functions

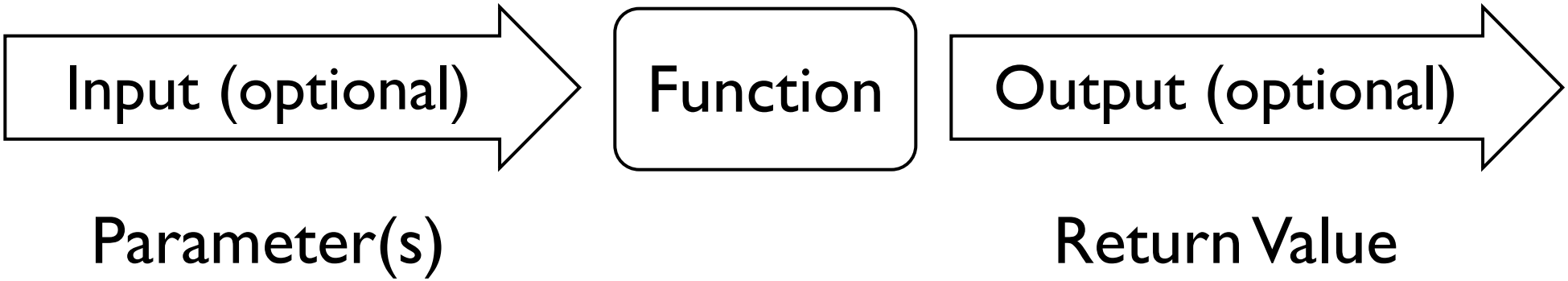
- Building blocks of programs
- Most code must be inside a function
- Programming involves writing and using functions
  - C comes with a bunch of them
- Functions can receive and/or return information

Parameter(s)

Return Value

Functions

- We've seen three functions so far
  - `main()`
  - `printf()`
  - `scanf()`



Parameter(s)

Return Value

with these parameters

```
printf("Hello\n");
```

name of function

call it

Defining Functions

```
return-type identifier(parameters)
    statement
```

```
int main(void)
{
    ...
}
```

Defining Functions

- *return-type*
  - Any type (e.g., `int`, `double`, `bool`, etc.), but not arrays
  - `void`
- *identifier*
  - Same rules as variable naming, but must be unique
- *parameters*
  - List of variable types and names
  - Any type, including arrays
  - `void`

Defining Functions

- I'm defining a function called `main`
- It returns something of type `int`
- It takes `void` parameters (i.e., no parameters)

```
int main(void)
{
    ...

    return 0;
}
```

Defining Functions

- I'm defining a function called `average`
- It returns something of type `double`
- It takes two parameters
  - 1) double named `a` and 2) `double` named `b`

```
double average(double a, double b)
{
   return (a + b) / 2;
}
```

Defining Functions

- I'm defining a function called `average`
- It returns something of type `double`
- It takes two parameters
  - 1) double named `a` and 2) `double` named `b`

```
double average(double a, double b)
{
   return (a + b) / 2;
}
```

```
{ double a;
{ double b;
```

Using Functions

```
double average(double a, double b)

int getSpeed(void)

void printSquared(int n)

void printHello(void)
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}

double x = 5.0;
double y = 2.0;

double avg = average(x, y);
```

avg: 

Functions w/ Return Types

```
                                        5.0
            double average(double a, double b)
            {double a;
            {double b;
                return (a + b) / 2;
            }
double x = 5.0;
double y = 2.0;                                  avg:
```

```
double avg = average(x, y);
```

Functions w/ Return Types

```
                                        5.0        2.0
                double average(double a, double b)
                {double a;
                {double b;
                    return (a + b) / 2;
                }
double x = 5.0;
double y = 2.0;                                  avg:
```

```
double avg = average(x, y);
```

Functions w/ Return Types

```
                                        5.0        2.0
                 double average(double a, double b)
                 {double a;
a = 5.0;         {double b;
b = 2.0;  ⟶          return (a + b) / 2;
                 }

double x = 5.0;
double y = 2.0;                                  avg:

double avg = average(x, y);
```

Functions w/ Return Types

```
                                        5.0        2.0
                 double average(double a, double b)
                 {double a;
a = 5.0;         {double b;
b = 2.0;  ------>    return (a + b) / 2;
                 }                    3.5

double x = 5.0;
double y = 2.0;                                  avg:

double avg = average(x, y);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

3.5

```
double x = 5.0;
double y = 2.0;

double avg = average(x, y);
```

avg: 

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}

double x = 5.0;
double y = 2.0;                    avg:
                3.5
double avg = average(x, y);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}

double x = 5.0;
double y = 2.0;                    avg: 3.5
              3.5
double avg = average(x, y);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}

double x = 5.0;
double y = 2.0;

double avg = average(x, y);
```

avg: 3.5

Functions w/ Return Types

```
                                        1.0
                double average(double a, double b)
                {double a;
                {double b;
                    return (a + b) / 2;
                }
double x = 5.0;
double y = 2.0;                                  avg: 3.5

double avg = average(x, y);

avg = average(1.0, 19.0);
```

Functions w/ Return Types

```
                                        1.0       19.0
                double average(double a, double b)
                {double a;
                {double b;
                    return (a + b) / 2;
                }
double x = 5.0;
double y = 2.0;                                  avg: 3.5

double avg = average(x, y);

avg = average(1.0, 19.0);
```

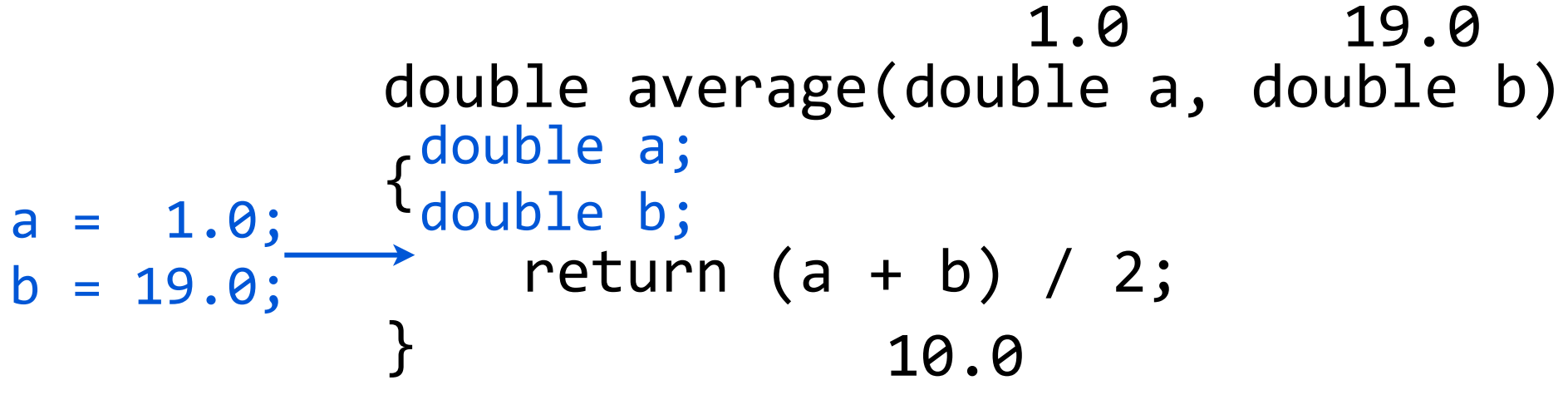
Functions w/ Return Types

```
                                        1.0       19.0
                 double average(double a, double b)
                 {double a;
a =  1.0;        {double b;
b = 19.0;  ⟶         return (a + b) / 2;
                 }
double x = 5.0;
double y = 2.0;                                  avg: 3.5

double avg = average(x, y);

avg = average(1.0, 19.0);
```

Functions w/ Return Types




```
double x = 5.0;
double y = 2.0;
```

avg: 3.5

```
double avg = average(x, y);

avg = average(1.0, 19.0);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}                          10.0
double x = 5.0;
double y = 2.0;                              avg: 3.5

double avg = average(x, y);

avg = average(1.0, 19.0);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}

double x = 5.0;
double y = 2.0;

double avg = average(x, y);
             10.0
avg = average(1.0, 19.0);
```

avg: 3.5

Functions w/ Return Types

```
double average(double a, double b)
{double a;
 double b;
    return (a + b) / 2;
}

double x = 5.0;
double y = 2.0;

double avg = average(x, y);
             10.0
avg = average(1.0, 19.0);
```

avg: 10.0

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

Functions w/ Return Types

```
                                        5.0
              double average(double a, double b)
              {double a;
              {double b;
                  return (a + b) / 2;
              }
double x = 5.0;
double y = 2.0;                                  avg: 10.0

double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

Functions w/ Return Types

```
                                        5.0       25.0
                double average(double a, double b)
                {double a;
                {double b;
                    return (a + b) / 2;
                }
double x = 5.0;
double y = 2.0;                                  avg: 10.0

double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

Functions w/ Return Types

```
                                        5.0       25.0
              double average(double a, double b)
               {double a;
a =  5.0;      {double b;
b = 25.0;  ---->    return (a + b) / 2;
               }
double x = 5.0;
double y = 2.0;                                  avg: 10.0

double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

Functions w/ Return Types

```
                                        5.0       25.0
                double average(double a, double b)
                {double a;
a =  5.0;       {double b;
b = 25.0;  ---->    return (a + b) / 2;
                }                  15.0
double x = 5.0;
double y = 2.0;                                avg: 10.0

double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

15.0

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

15.0

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

5

15.0

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

5

Functions w/ Return Types

```
double average(double a, double b)
{double a;
double b;
    return (a + b) / 2;
}
```

```
double x = 5.0;
double y = 2.0;
```

avg: 10.0

```
double avg = average(x, y);

avg = average(1.0, 19.0);

printf("%g", average(5.0, 25.0) / 3);
```

5

```
average(x + 1, y + 1);
```

Legal, but useless `x + 5;`

```
average;
```

Legal, but not what you expect

Using void Functions

```
void printSquared(int n)
{
  printf("%d squared is %d", n, n * n);
}
```

```
printSquared(2);
int i = 5;
printSquared(i);

i = printSquared(9);
printf("%d", printSquared(3));
```

```
2 squared is 4
5 squared is 25
```

Error

Error

Side Effects

- `void` functions need a side effect
- Functions with return types may have side effects
- This means you will sometime (intentionally) ignore a return value

`numChars:` 21

```
int numChars = printf("I have a side effect.");
```

```
I have a side effect.
```

```
printf("%d", printf("I have a side effect."));
```

```
I have a side effect.21
```