Functions

Chapter 6, page 303

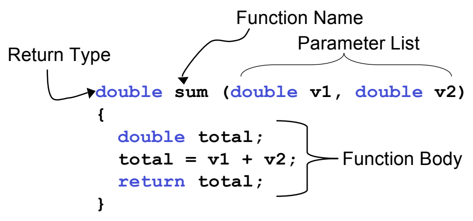
Functions

- “A collection of statements that perform a specific task”, p 303
  - And can be accessed at any point in the code through a *function call* and optionally produce a value

```
#include <cmath>

x = pow(2.0, 3);
x = pow(4.0, 2);
```

Functions



```
double sum (double v1, double v2)
{
  double total;
  total = v1 + v2;
  return total;
}
```

Functions

- Functions are a way of building *modules* in your program
- Encapsulate some calculation
- Less repetitive code
- Example:
  - `x = sum(4.0, 2.2);`
  - `cout << x << endl;`

Parts of the Function

- Function Name
- Function Body
- Parameter List
- Return Type

Calling a function

```
double sum (double v1, double v2)
{
  double total;
  total = v1 + v2;
  return total;
}

int main()
{
  double value1, value2;
  value1 = sum(4.2, 2.4);
  value2 = sum(value1, 2.4);
  cout << value2;
  return 0;
}
```

`void` Functions

- Not all functions need to produce a value
- These functions return `void`

```
void printDayOfWeek (int day)
{
  if ( day == 0 )
  {
    cout << " Sunday ";
  }
  else if (day == 1 )
  {
    cout << " Monday ";
  }
  . . .
  return;
}
```

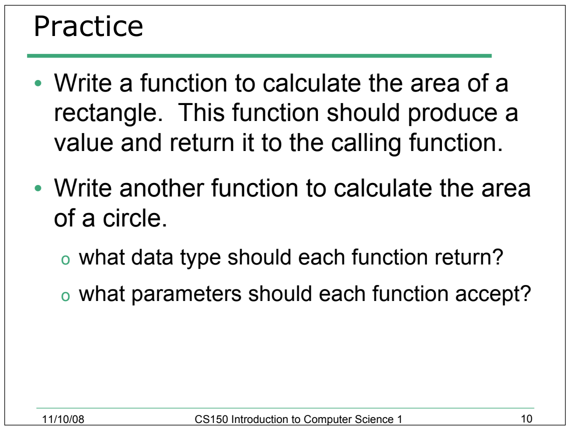
What will happen?

```
void printSquares (int value, int value2)
{
   cout << value * value << " ";
   cout << value2 * value2 << endl;
  return;
}

int main()
{
  int x = 3, y = 2;
  printSquares(x, y);
  printSquares(y, x);
  return 0;
}
```

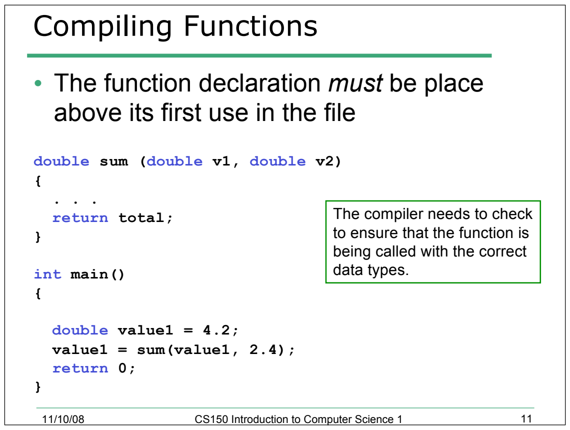
Practice

- Write a function that will calculate the average of three numbers and print the result to the screen.
- What parameters do you need?
- What should the return type be?



Practice

- Write a function to calculate the area of a rectangle. This function should produce a value and return it to the calling function.
- Write another function to calculate the area of a circle.
  - what data type should each function return?
  - what parameters should each function accept?



Compiling Functions

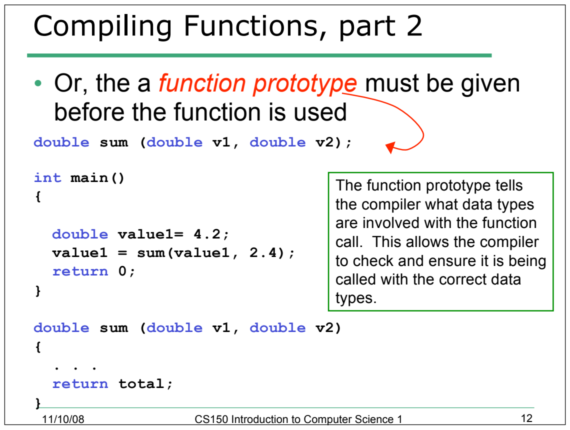
- The function declaration *must* be place above its first use in the file

```
double sum (double v1, double v2)
{
  . . .
  return total;
}

int main()
{

  double value1 = 4.2;
  value1 = sum(value1, 2.4);
  return 0;
}
```

The compiler needs to check to ensure that the function is being called with the correct data types.



Compiling Functions, part 2

- Or, the a *function prototype* must be given before the function is used

```
double sum (double v1, double v2);

int main()
{

  double value1= 4.2;
  value1 = sum(value1, 2.4);
  return 0;
}

double sum (double v1, double v2)
{
  . . .
  return total;
}
```

The function prototype tells the compiler what data types are involved with the function call. This allows the compiler to check and ensure it is being called with the correct data types.

What will happen?

```
void square (int value)
{
  cout << (value * value);
  return;
}

int main()
{
  for(int i = 0; i< 5; i++)
  {
    cout << i << ": ";
    square(i);
    cout << endl;
  }
  return 0;
}
```

```
int square (int value);

int main()
{
  for(int i = 0; i< 5; i++)
  {
    cout << i << ": ";
    cout << square(i);
    cout << endl;
  }
  return 0;
}

int square (int value)
{
  return (value * value);
}
```

bool return values

```
bool isEven (int value)
{
  return (value % 2)==0;
}

int main()
{
  int x = 9, y = 10;
  if( isEven(x) )
  {
    cout << "EVEN: " << x << endl;
  }
  if( isEven(y) )
  {
    cout << "EVEN: " << y << endl;
  }
  return 0;
}
```

Practice

- Build a small program that asks the user for either a rectangle or circle and displays the area of the selection shape. Use the functions we just defined.
- Continue asking for input until the user types something other than ‘r’ or ‘c’.

Passing Arguments

- Arguments are passed into functions
- Parameters are evaluated in the order given
- A **copy** of the argument is made in the parameter
- If a parameter is changed in the function, is that reflected in main?

What will happen?

```
void swap (int value, int value2)
{
  int tmp = value;
  value = value2;
  value2 = tmp;
  cout << value << " " << value2 << endl;
  return;
}

int main()
{
  int x = 9, y = 10;
  swap(x, y);
  cout << x << " --- " << y << endl;
  return 0;
}
```

parameters

arguments

Passing Arguments

- Pass by value
  - arguments are **copied** into the parameter list
  - changes made in the function will **not** be reflected in main
- Pass by reference
  - changes made in the function are reflected in the main

Example

```
void swap(int &, int &);
int main(void)
{
  int i, j;
  cin >> i >> j;
  swap(i,j);
  cout << i << j;
  return 0;
}

void swap(int & num1, int & num2)
{
  int temp;
  temp = num1;
  num1 = num2;
  num2 = temp;
  return;
}
```

Rules for Parameter Lists

- Same number of arguments as parameters
- Arguments & parameters are matched by position
- Arguments & parameters must have the same type
- The names of the arguments and parameters may be the same or different
- For reference parameters only, the parameter must be a single, simple variable

Example

- Given the following function prototype:

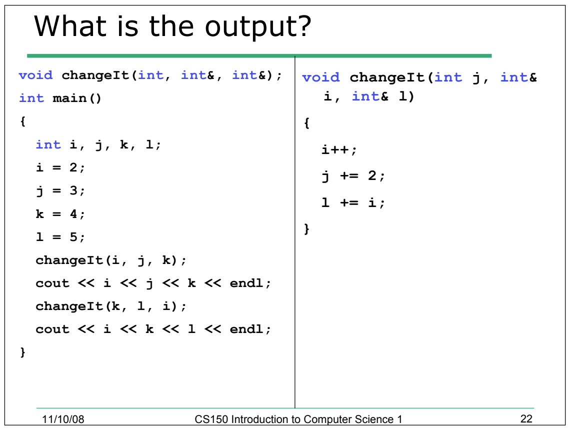
```
void checkIt(float &, float &, int, int, char &);
```

- And declarations in main:

```
float x, y;
int m;
char next;
```

Which are legal?

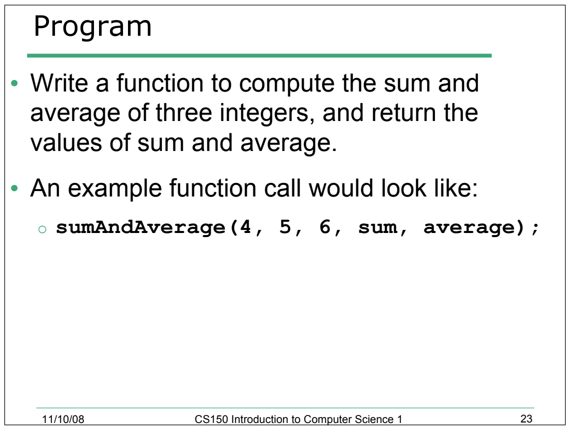
```
checkIt(x, y, m+3, 10, next);
checkIt(m, x, 30, 10, 'c');
checkIt(x, y, m, 10);
checkIt(35.0, y, m, 12, next);
checkIt(x, y, m, m, c);
```



What is the output?

```
void changeIt(int, int&, int&);
int main()
{
  int i, j, k, l;
  i = 2;
  j = 3;
  k = 4;
  l = 5;
  changeIt(i, j, k);
  cout << i << j << k << endl;
  changeIt(k, l, i);
  cout << i << k << l << endl;
}
```

```
void changeIt(int j, int&
  i, int& l)
{
  i++;
  j += 2;
  l += i;
}
```



Program

- Write a function to compute the sum and average of three integers, and return the values of sum and average.
- An example function call would look like:
  - `sumAndAverage(4, 5, 6, sum, average);`