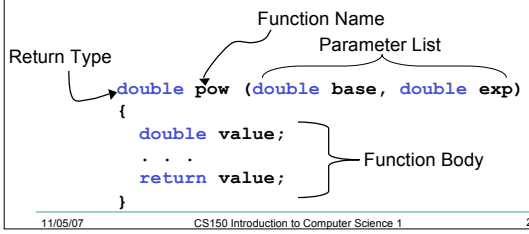
Functions

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



Functions

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

Parts of the Function

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

Calling a function

```
double pow (double base, double exp)
{
  double value;
  . . .
  return value;
}

int main()
{

  double value1, value2;
  value1 = pow(4.2, 2.4);
  value2 = pow(value1, 2.4);
  return 0;
}
```


`void` Functions

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

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

Q.1 Practice

- Write a function that will display a menu to the screen.

```
Where were you born?
1. California
2. Hawaii
3. Oregon
4. Washington
5. West Virginia
6. Wyoming
7. Some Other Place
```

- What should the return type be?
- What parameters do you need?

Q.2 Practice

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

Compiling Functions

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

```
double pow (double base, double exp)
{
  . . .
  return value;
}

int main()
{

  double value1 = 4.2;
  value1 = pow(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 pow (double base, double exp);

int main()
{
  double value1= 4.2;
  value1 = pow(value1, 2.4);
  return 0;
}

double pow (double base, double exp)
{
  . . .
  return value;
}
```

The function definition 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.

Q.3 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);
}
```


Q.4 What will happen?

```
void swap (int value, int value2)
{
  int tmp = value;
  value = value2;
  value2 = tmp;
  return;
}

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

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;
}
```


Q.5 Practice

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

Q.6 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’.