Example: function calls function

```c
void deeper() {
    cout << "I am now in function deeper.\n";
}
void deep() {
    cout << "Hello from the function deep.\n";
    deeper();
    cout << "Back in function deep.\n";
}
int main() {
    cout << "Hello from Main.\n";
    deep();
    cout << "Back in function Main again.\n";
    return 0;
}
```

Output:
Hello from Main.
Hello from the function deep.
I am now in function deeper.
Back in function deep.
Back in function Main again.
Example: call func more than once

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

void pluses(int count) {
    for (int i = 0; i < count; i++)
        cout << "+";
    cout << endl;
}

int main() {
    int x = 2;
    pluses(4);
    pluses(x);
    pluses(x+5);
    pluses(pow(x, 3.0));
    return 0;
}
```

Output:
```
++++
++
+++++++
```

Example: multiple parameters

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

void pluses(char ch, int count) {
    for (int i = 0; i < count; i++)
        cout << ch;
    cout << endl;
}

int main() {
    int x = 2;
    char cc = '!';
    pluses('#', 4);
    pluses('*', x);
    pluses(cc, x + 5);
    pluses('x', pow(x, 3.0));
    return 0;
}
```

Output:
```
####
**
!!!!!!!
```

xxxxxxx
Passing Arguments by Value

- **Pass by value**: when an argument is passed to a function, its value is *copied* into the parameter.
- Parameter passing is implemented using variable initialization: `int param = argument;`
- Changes to the parameter in the function do not affect the value of the argument

Example: Pass by Value

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

void changeMe(int);

int main() {
    int number = 12;
    cout << "number is " << number << endl;
    changeMe(number);
    cout << "Back in main, number is " << number << endl;
    return 0;
}

void changeMe(int myValue) {
    myValue = 200;
    cout << "myValue is " << myValue << endl;
}
```

Output:
- number is 12
- myValue is 200
- Back in main, number is 12
Pass by Value

- Parameter is initialized to a copy of the argument’s value.

- Even if the body of the function changes the parameter, the argument in the calling function is unchanged.

- The parameter and the argument are stored in separate variables, separate locations in memory.

The return statement

- Used to end execution of a function
- Can be placed anywhere in a function--the function will transfer control back to where it was called from immediately.
- Statements that follow the return statement will not be executed
  - unless return is in an if-branch
- In a void function without a return statement, there is an implicit return statement before the last }
return: example

```c++
void someFunc (int x) {
    if (x < 0) {
        cout << "x must not be negative." << endl;
    } else {
        // Continue with lots of statements, indented
        // ...
        // so many it’s hard to keep track of matching {} 
    }
}

The following function is equivalent, easier to read:

```c++
void someFunc (int x) {
    if (x < 0) {
        cout << "x must not be negative." << endl;
        return;
    }  
    // Continue with lots and lots of statements
    // ...
}
```

return: don’t do this

- The cout will never happen . . . ever

```c++
void someFunc (int x) {
    if (x < 0) {
        return;
        cout << "x must not be negative." << endl;
    } 
    // Continue with lots and lots of statements
    // ...
}
```
Returning a value from a function

- You can use the return statement to send a value back to the function call.
  
  \[ \text{return expr;} \]

- The value of the expr will be sent back.

- The data type of the value the function is returning is required in the function header:

  \[
  \text{int doubleIt}(\text{int} \ x) \ \{ \\
  \text{return} \ x*2; \\
  \} \\
  \]

Calling a function that returns a value

- If the function returns void, the function call is a statement:

  \[ \text{pluses}(4); \]

- If the function returns a value, the function call is an expression:

  \[ \text{int} \ x = \text{doubleIt}(4); \]

- The value of the function call expression is the value of the expr returned from the function.
# Returning the sum of two ints

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

int sum(int, int);

int main() {
    int value1;
    int value2;
    int total;
    cout << "Enter 2 numbers: " << endl;
    cin >> value1 >> value2;
    total = sum(value1, value2);
    cout << "The sum is " << total << endl;
    return 0;
}

int sum(int x, int y) {
    return x + y;
}
```

Output:
```
Enter 2 numbers: 
12 45
The sum is 57
```

## Data transfer

- **The function call from main:**
  
  ```
  total = sum(value1, value2);
  ```

- **The arguments are passed in to the function:**
  ```
  int x = value1;
  int y = value2;
  ```

- **The result, \(x+y\), is returned to the call.**
  ```
  total = sum(value1, value2);
  ```
  ```
  The value is the result of \(x + y\)
  ```
Function call expression

- When the function returns a value, the function call is an expression.
- The function call can occur in any context where an expression (of the return type) is allowed:
  - assign to variable (or array element)
  - output via cout
  - use in a more complicated expression
  - pass as an argument to another function
- The value of the function call expr is determined by the value of the expression returned from the function.

Calculating the area of a circle

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

double getRadius();
double square(double);

int main() {
    const double PI = 3.14159;
    double radius;
    double area;
    cout << fixed << setprecision(2);
    radius = getRadius();
    area = PI * square(radius);
    cout << "The area is " << area << endl;
    return 0;
}
```

This works here too:

```cpp
area = PI * square(getRadius());
```
Calculating the area of a circle

```cpp
double getRadius() {
    double rad;
    cout << "Enter the radius of the circle: ";
    cin >> rad;
    return rad;
}

double square(double number) {
    return number * number;
}
```

Output:
Enter the radius of the circle: 1.2
The area is 4.52

You can use a function that returns a value to input a value and return it to the main function.

Returning a boolean value

```cpp
bool isValid(int number)
{
    bool status;
    if (number >= 1 && number <= 100)
        status = true;
    else
        status = false;
    return status;
}
```

- the above function is equivalent to this one:

```cpp
bool isValid (int number) {
    return (number >= 1 && number <= 100);
}
```
Returning a boolean value

• You can use the function as follows:

    bool isValid(int);

    int main() {
        int val;
        cout << “Enter a value between 1 and 100: ”
        cin >> val;

        while (!isValid(val)) {
            cout << “That value was not in range.\n”;  
            cout << “Enter a value between 1 and 100: ”
            cin >> val;
        }
        // . . .