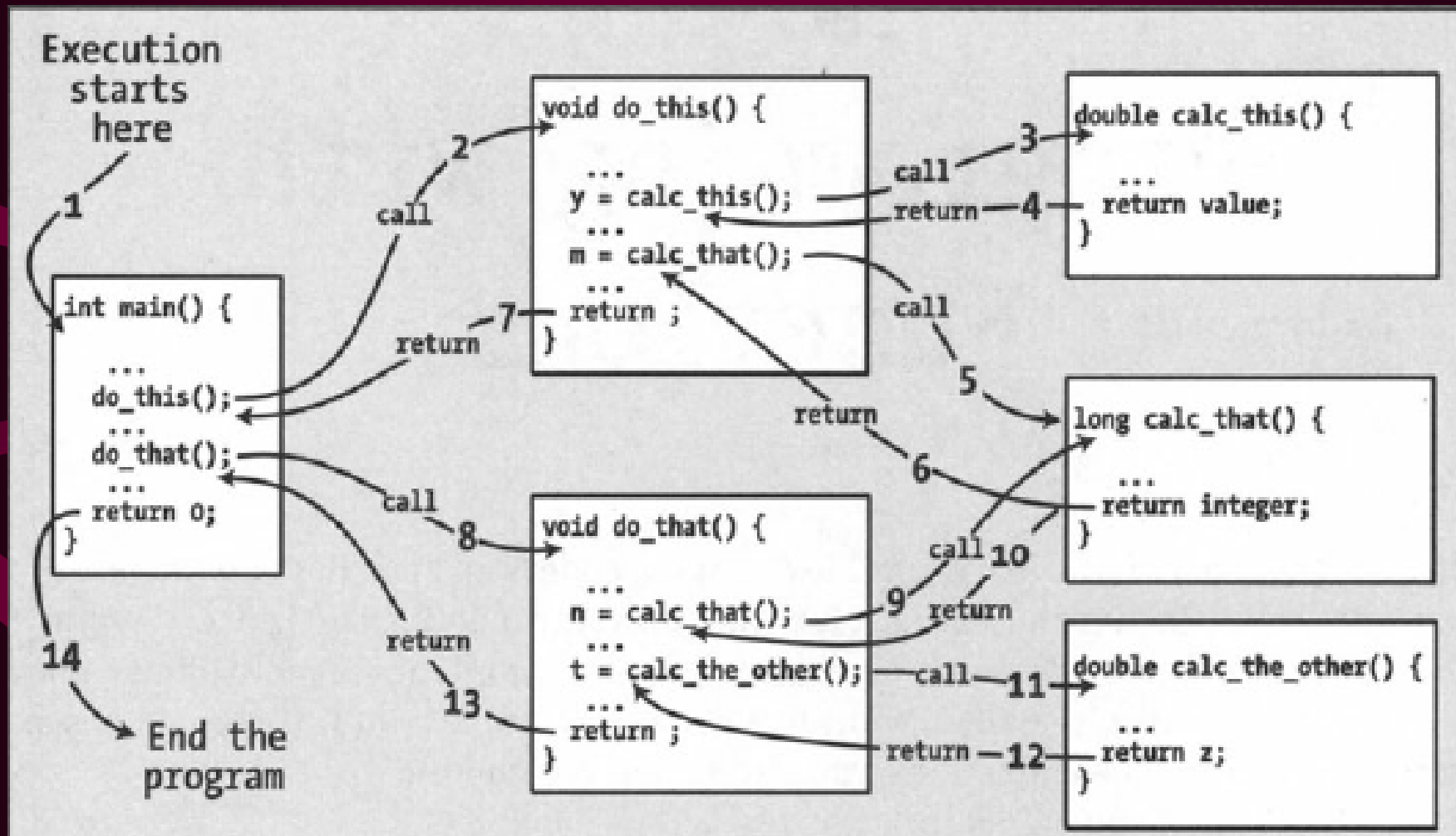


# Functions

- Reasons
- Concepts
- Passing arguments to a function
- Preset parameters
- Function returns
- Inline functions
- Static variables within a function

# Reasons



# Concepts

`cout << power( 3.0 , 2 );` The arguments in a function call map to the parameters in its definition.

The value 9.0 is returned to the calling function when the function completes execution. This return value will then be used within the expression in which the function call appeared.

```
double power( double x, int n) {  
    double result = 1.0;  
    if(n >= 0)  
        for(int i = 1 ; i <= n ; i++)  
            result *= x;  
    else  
        for(int i = 1 ; i <= -n ; i++)  
            result /= x;  
    return result;  
}
```

When the code in the body of the function executes, the parameters will have values corresponding to the arguments that appeared in the function call statement.

name and its parameter list.

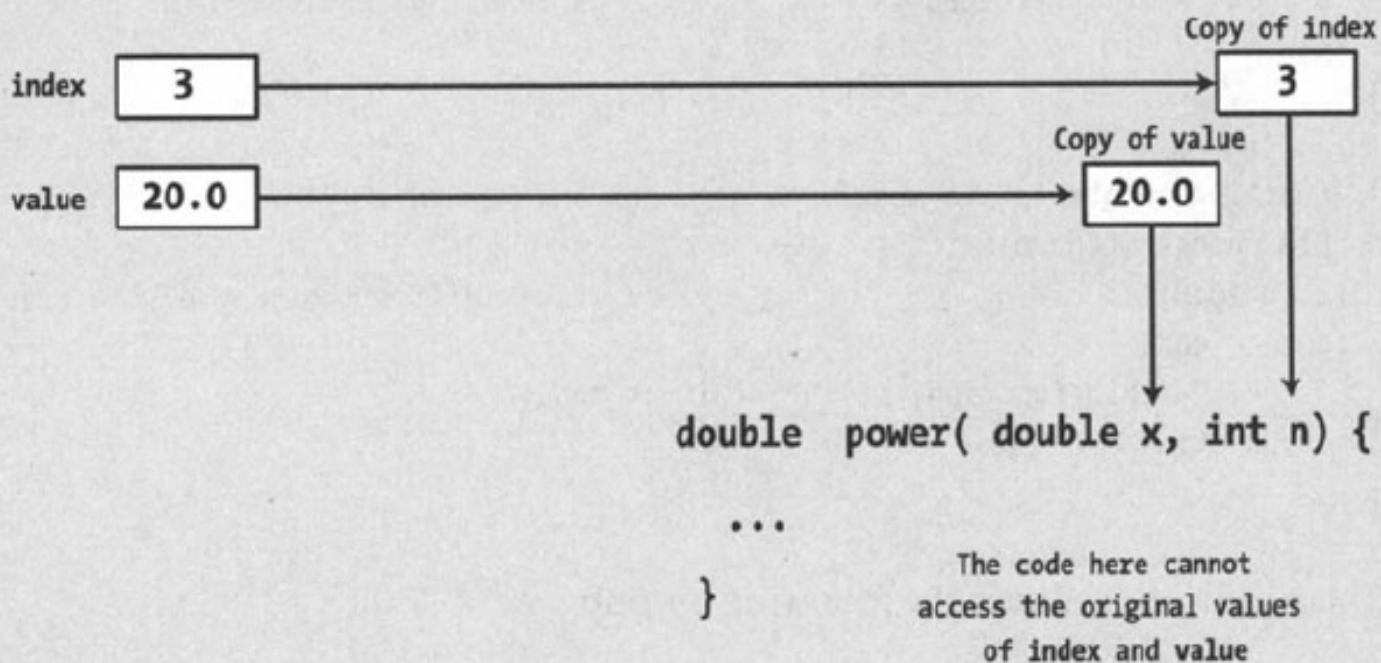
# Function Headers

- `return_type function_name(parameter_list);`
  - Data return by `return_type`, e.g.
  - `double sine(double t); // prototype`
  - `double d = sine(3.5); // calling`
- `void function_name(parameter_list);`
  - Data passing and return by `parameter_list`, e.g.
  - `void add(int x, int y, int &z); // prototype`
  - `add(1, 2, k); cout << k; // 3, calling`

# Passing arguments to a Function

- Pass-by-value (program 8.3)

```
double value = 20.0;  
int index = 3;  
double result = power(value, index);
```



# Passing a pointer

- Passing a pointer (Program 8.4)
- Passing an array as a function argument (Program 8.5)
- Using pointer notation when passing array (program 8.6)
- Constant pointer parameters
- Passing a multi-dimension array to a function (Program 8.7)

# Pass-by-reference

- Using a reference parameter (Program 8.8)

- Calling Function

## Function with a Reference Parameter

- Function with a Pointer Parameter

```
int change_it(int* value) {  
    *value += 10;  
    return *value;  
}
```

When you call this function, the argument must be an address. In this case, it is obvious from the call that the following function might be able to alter the argument's value:

```
int number = 20;  
cout << change_it(&number);
```

## Function with a Reference Parameter

```
int change_it(int& value) {  
    value += 10;  
    return value;  
}
```

Here the argument is passed in the form of a reference. The function call doesn't indicate that it is possible for the following function to alter the value of the argument:

```
int number = 20;  
cout << change_it(number);
```



# References vs Pointers

- In most situations, using a reference parameter is preferable to using a pointer
- An important difference between a pointer and a reference is that a pointer can be null, whereas a reference always refers to something



# Arguments to main()

- `int main(int argc, char* argv[])`

```
{ // comment
    // cout
    return 0;
}
```

```
// Program that lists its command line arguments
#include <iostream>
using std::cout;
using std::endl;
```

- Command

```
int main(int argc, char* argv[]) {
    for (int i = 0 ; i < argc ; i++)
        cout << endl << argv[i];

    cout << endl;
    return 0;
}
```

# Multiple Default Argument Values

- Program 8.9

# Returning Values from a Function

- Returning a value
- Returning a pointer
  - Never return the address of an automatic local variable from a function

- `*larger(value1, value2) = 100;`

```
int* larger(int a, int b) {  
    if(a > b)  
        return &a;           // Wrong!  
    else  
        return &b;           // Wrong!  
}
```

automatic

# Returning a pointer

- Variables
  - Local variables
  - Global variables
  - Static variables

```
*longer(&value1, &value2) = 100;  
int* larger(int*a, int *b) {  
    return *a>*b ? a:b; } // i.e. a =100 or b=100
```

- Program 8.10

# Returning a reference

```
larger(value1, value2) = 50;  
int& larger(int& m, int& n) {  
    return m > n ? m : n;  
}
```

# Inline functions

- Very short function definitions
- More efficient and fast execution

```
inline int larger(int m, int n)
{ return m > n ? m : n; }
```

# Static Variables

- Static variable is created and remains in existence until the program terminates. One can carry over a value from one call of a function to the next.
- Program 8.11