

Executing a Program

- Make space (boxes) for any global variables (variables that do not appear inside a function definition)
- Make space for any variables in the function main
- Execute the statements in function main (in order from the first statement after the { to the last statement before the })

Executing a Function

- Every time you encounter a function call you should follow a specific set of steps to execute that function
- To do this, we need to identify certain parts of a program:
 - function definition
 - return type
 - function name
 - parameters
 - function body
 - local variables
 - local statements
 - function call
 - function name
 - arguments

Parts of a Function Definition

```

Return type
/
Function name Parameters
void ShowCost(int quarts, int dozens, float cost)
{
float gallons;
int cookies;
}
Function definition
gallons = quarts / 4.0;
cookies = dozens * 12;
printf("Your total is $%.2f for %.2f gallons of
milk and %d cookies.\n", cost, gallons,
cookies);
}
    
```

Local variables

Statements in body

Function Calls

```

void main() {
int milk1 = 4;
int cookie1 = 5;
int milk2 = 3;
int cookie2 = 6;

ShowCost(milk1, cookie1, 14.00);
ShowCost(milk2, cookie2, 15.00);
}
    
```

Function calls

Arguments

Executing a Function Call

1. Setup the parameters
 - a. Evaluate the arguments in the call
 - b. Make boxes labeled with the parameter names
 - c. Copy the values in order from the arguments to the parameters
2. Setup other variables
 - a. If the return type is something other than void make a box labeled ReturnOfFunctionName
 - b. Make boxes for the local variables
3. Execute the statements in the function from top to bottom

Stop when:

 - a. A return statement is encountered (calculate the value following the return and put that value in the box labeled ReturnOfFunctionName)
 - b. The } is encountered
4. Cleanup

Eliminate the boxes for the parameters and the local variables
5. Go back to where the call was made

If the call was part of a statement, the value calculated by the function is in the box labeled ReturnOfFunctionName (once you've used this box you can delete it)

Executing a Function

```

void ShowCost(int quarts, int dozens, float cost) {
float gallons;
int cookies;

gallons = quarts / 4.0;
cookies = dozens * 12;

printf("Your total is $%.2f for %.2f gallons of milk
and %d cookies.\n", cost, gallons, cookies);
}

void main() {
int milk1 = 4;
int cookie1 = 5;
int milk2 = 3;
int cookie2 = 6;

ShowCost(milk1, cookie1, 14.00);
ShowCost(milk2, cookie2, 15.00);
}
    
```

| | |
|-------|---------|
| milk1 | cookie1 |
| 4 | 5 |
| milk2 | cookie2 |
| 3 | 6 |

Start here

1a. Evaluate Arguments

```

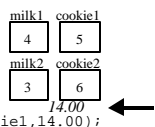
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of milk
    and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    ShowCost(milk1, cookie1, 14.00);
    ShowCost(milk2, cookie2, 15.00);
}

```



1b. Make Boxes for Parameters

```

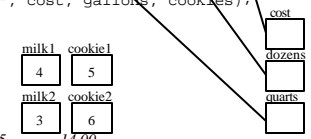
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of milk
    and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    ShowCost(milk1, cookie1, 14.00);
    ShowCost(milk2, cookie2, 15.00);
}

```



1c. Copy argument values to corresponding parameters

```

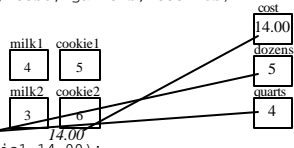
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of milk
    and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    ShowCost(milk1, cookie1, 14.00);
    ShowCost(milk2, cookie2, 15.00);
}

```



2b. (skip 2a, return type void) Make boxes for local variables

```

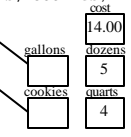
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of
    milk and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    ShowCost(milk1, cookie1, 14.00);
    ShowCost(milk2, cookie2, 15.00);
}

```



3. Execute the statements in the body of the function

```

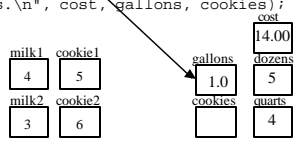
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of
    milk and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    ShowCost(milk1, cookie1, 14.00);
    ShowCost(milk2, cookie2, 15.00);
}

```



3. Execute the statements in the body of the function

```

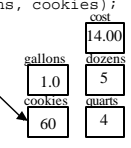
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of
    milk and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    ShowCost(milk1, cookie1, 14.00);
    ShowCost(milk2, cookie2, 15.00);
}

```



3. Execute the statements in the body of the function

```
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

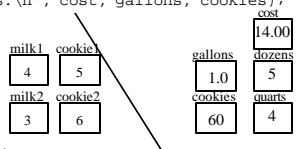
    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of
    milk and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;

    ShowCost (milk1,cookie1,14.00);
    ShowCost (milk2,cookie2,15.00);
}

```



Output:
Your total is \$14.00 ...

4. Cleanup - eliminate boxes for local variables, parameters

```
void ShowCost(int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

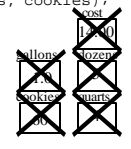
    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of
    milk and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;

    ShowCost (milk1,cookie1,14.00);
    ShowCost (milk2,cookie2,15.00);
}

```



5. Go back to where the call was made

```
void ShowCost (int quarts, int dozens, float cost) {
    float gallons;
    int cookies;

    gallons = quarts / 4.0;
    cookies = dozens * 12;

    printf("Your total is $%.2f for %.2f gallons of
    milk and %d cookies.\n", cost, gallons, cookies);
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;

    ShowCost (milk1,cookie1,14.00);
    ShowCost (milk2,cookie2,15.00);
}

```



← Now here, what next?

Functions with Return Types

- Functions that calculate or return value give us a mechanism for having a function produce one piece of information
- These functions generally have all value parameters, and always have a non-void return type

A Function Returning a Value

```
float CalcCost(int quarts, int dozens) {
    float milk_cost;
    float cookie_cost;

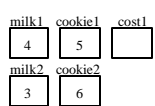
    milk_cost = quarts * 1.00;
    cookie_cost = dozens * 2.00;

    return milk_cost + cookie_cost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1,cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2,cookie2));
}

```



← Start here

1. Setup the parameters

```
float CalcCost(int quarts, int dozens) {
    float milk_cost;
    float cookie_cost;

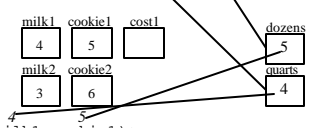
    milk_cost = quarts * 1.00;
    cookie_cost = dozens * 2.00;

    return milk_cost + cookie_cost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1,cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2,cookie2));
}

```

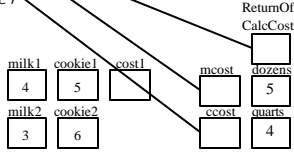


2. Setup other variables

```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;
    mcost = quarts * 1.00;
    ccost = dozens * 2.00;
    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```

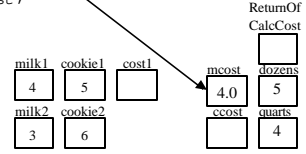


3. Execute the statements in the body of the function

```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;
    mcost = quarts * 1.00;
    ccost = dozens * 2.00;
    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```

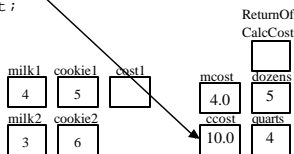


3. Execute the statements in the body of the function

```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;
    mcost = quarts * 1.00;
    ccost = dozens * 2.00;
    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```

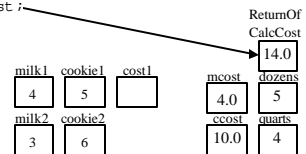


3. Execute the statements in the body of the function

```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;
    mcost = quarts * 1.00;
    ccost = dozens * 2.00;
    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```

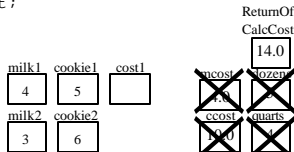


4. Cleanup, eliminate boxes for local variables, parameters

```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;
    mcost = quarts * 1.00;
    ccost = dozens * 2.00;
    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```

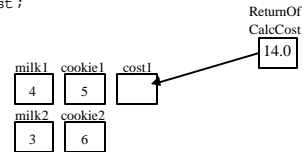


5. Go back to where call was made, return value is in box ReturnOf...

```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;
    mcost = quarts * 1.00;
    ccost = dozens * 2.00;
    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```



5. Once value is used you can eliminate ReturnOf ... box

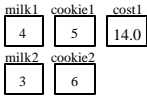
```
float CalcCost(int quarts, int dozens) {
    float mcost;
    float ccost;

    mcost = quarts * 1.00;
    ccost = dozens * 2.00;

    return mcost + ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1;

    cost1 = CalcCost(milk1, cookie1);
    printf("Cost 2 is $%.2f\n", CalcCost(milk2, cookie2));
}
```



Now what??

Reference Parameters

- Reference parameters give us a mechanism for getting more than one piece of information out of a function
- To make a parameter a reference parameter we put a * in front of the name in the parameter list, and an & in front of the variable name in the argument list

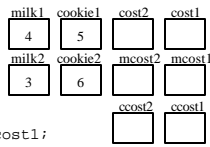
A Function with Reference Parameters

```
void CalcTheCosts(int quarts, int dozens, float *cost,
    float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;

    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```



Start here

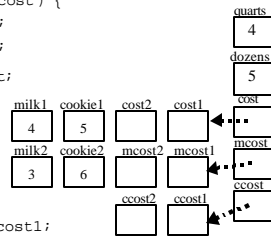
1. Setup the parameters

```
void CalcTheCosts(int quarts, int dozens, float *cost,
    float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;

    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```



Reference Parameters

- Each reference parameter stores the address of the variable that is passed as an argument to the parameter
- Rather than showing an address (which only the program would actually know), we envision this connection as a pointer
- In the function, using the name of a parameter (as in cost) refers to the pointer and * followed by the name (as in *cost) refers to the box it points at

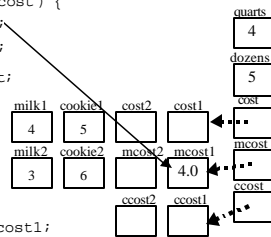
3. (Nothing happens in Step 2) Execute the body of the function

```
void CalcTheCosts(int quarts, int dozens, float *cost,
    float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;

    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```

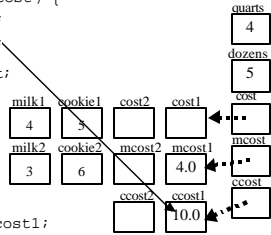


3. Execute the body of the function

```
void CalcTheCosts(int quarts, int dozens, float *cost,
float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;
    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```

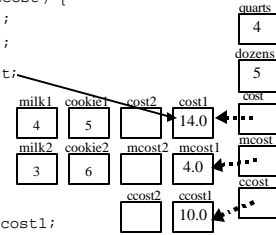


3. Execute the body of the function

```
void CalcTheCosts(int quarts, int dozens, float *cost,
float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;
    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```

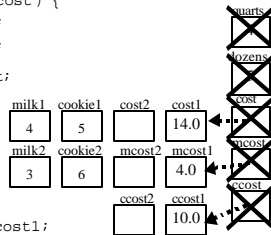


4. Cleanup, eliminate boxes for local variables, parameters

```
void CalcTheCosts(int quarts, int dozens, float *cost,
float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;
    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```

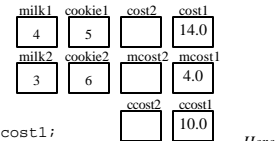


5. Go back to where call was made

```
void CalcTheCosts(int quarts, int dozens, float *cost,
float *mcost, float *ccost) {
    *mcost = quarts * 1.00;
    *ccost = dozens * 2.00;
    *cost = *mcost + *ccost;
}

void main() {
    int milk1 = 4;
    int cookie1 = 5;
    int milk2 = 3;
    int cookie2 = 6;
    float cost1, mcost1, ccost1;
    float cost2, mcost2, ccost2;

    CalcTheCosts(milk1, cookie1, &cost1, &mcost1, &ccost1);
    CalcTheCosts(milk2, cookie2, &cost2, &mcost2, &ccost2);
}
```



Here,
what next?