Module 5 Guidance Notes

Functions

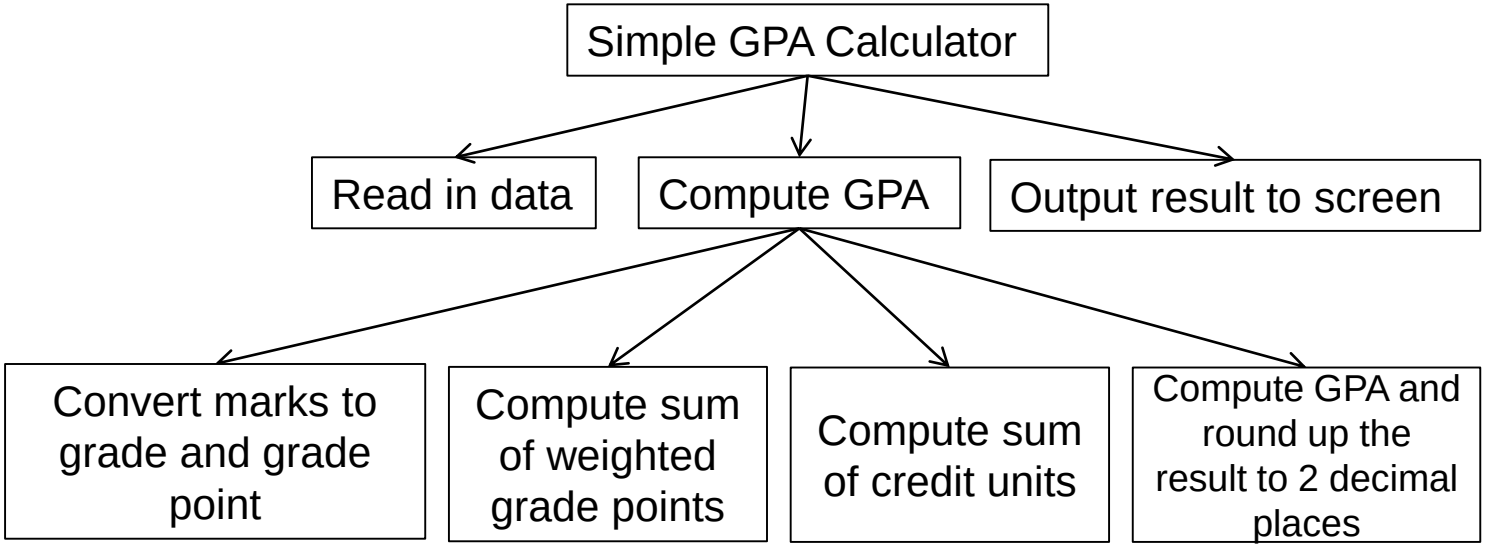
Estimated Time of Completion: 2.5 Hours

Outline

TOP-DOWN DESIGN (DIVIDE AND CONQUER) APPROACH

Top-Down Program Design

Top-Down Design



Each module should perform a single, well-defined task

Functions

```
main()
```

Advantages of Using Functions

```
main()
```

PREDEFINED FUNCTIONS VS. SELF-DEFINED FUNCTIONS

Predefined Functions

```
double x = sqrt(5.29);
```

Here, 5.29 is the function input and the function output 2.3 would be stored to x.

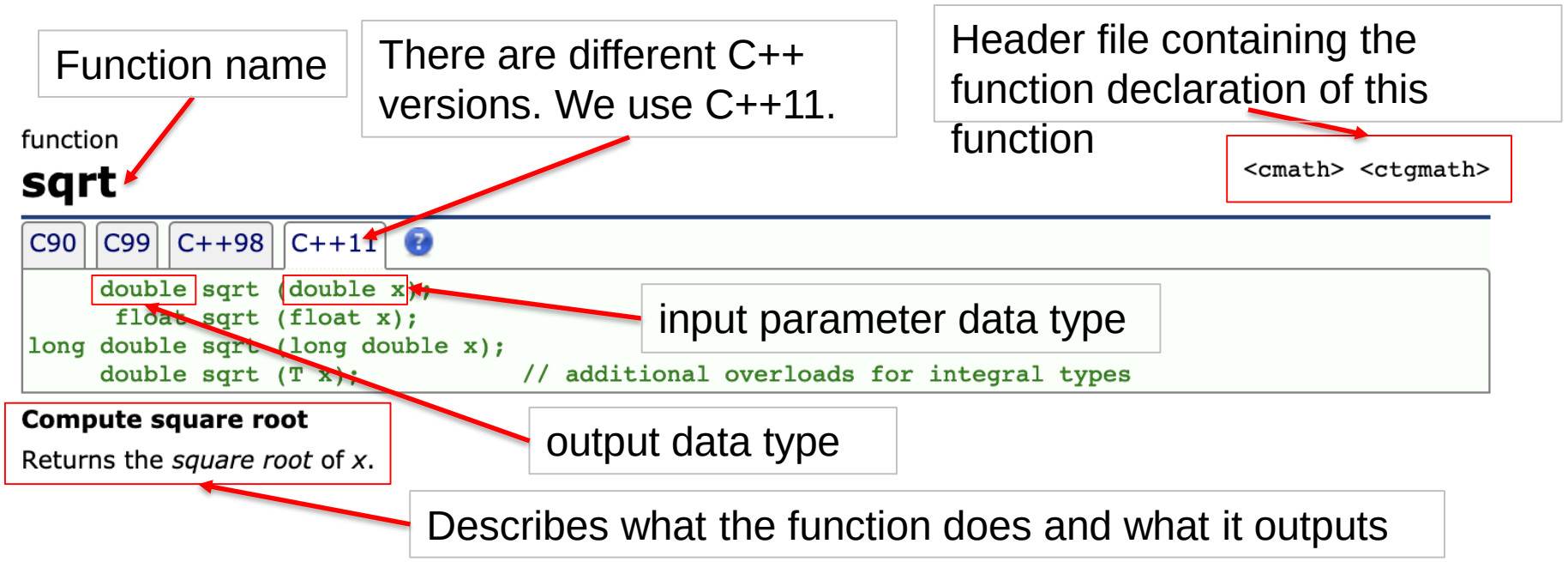
Predefined Functions

Examples

double sqrt(double x)		cmath
double pow(double x, double y)		cmath
double fabs(double x)		cmath
double ceil(double x)		cmath
double floor(double x)		cmath
int abs(int x)		cstdlib
int rand()		cstdlib

Using Predefined Functions

Function Reference Example



Using Predefined Functions

#include <...>

```
#include<iostream>
#include<cstdlib>
```

cin cout endl

rand() srand()

Using Predefined Functions

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

int main() {
    // Compute the root mean square of 10 input numbers
    int i;
    double n, sq_sum = 0;

    for(i=0; i<10; i++)
    {
        cout << i+1 << ": ";
        cin >> n;
        sq_sum += pow(n, 2.0);
    }

    cout << "The root mean square is " << sqrt(sq_sum/10) << endl;

    return 0;
}
```

Include <cmath> so you can use the pow() and sqrt() functions from the math library later on in the same program file.

A function may accept one or more input parameters. The order and type of each parameter matter. Check the pow() reference page to see what each parameter mean.

Example: Random Number Generation

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

int main()
{
  int num = 23;
  int guess;
  bool isGuessed;

  isGuessed = false;

  while (!isGuessed) {
    cout << "Make a guess (0-99)? ";
    cin >> guess;

    if (guess == num) {
      cout << "Correct!" << endl;
      isGuessed = true;
    }
    else if (guess < num)
      cout << "Too small.  Guess again!" << endl;
    else
      cout << "Too large.  Guess again!" << endl;
  }
  return 0;
}
```

Example: Random Number Generation

function

rand

<cstdlib>

```
int rand (void);
```

Generate random number

Returns a pseudo-random integral number in the range between 0 and RAND_MAX.

This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function srand.

RAND_MAX is a constant defined in <cstdlib>.

Example: Random Number Generation

```
rand() % 10
```

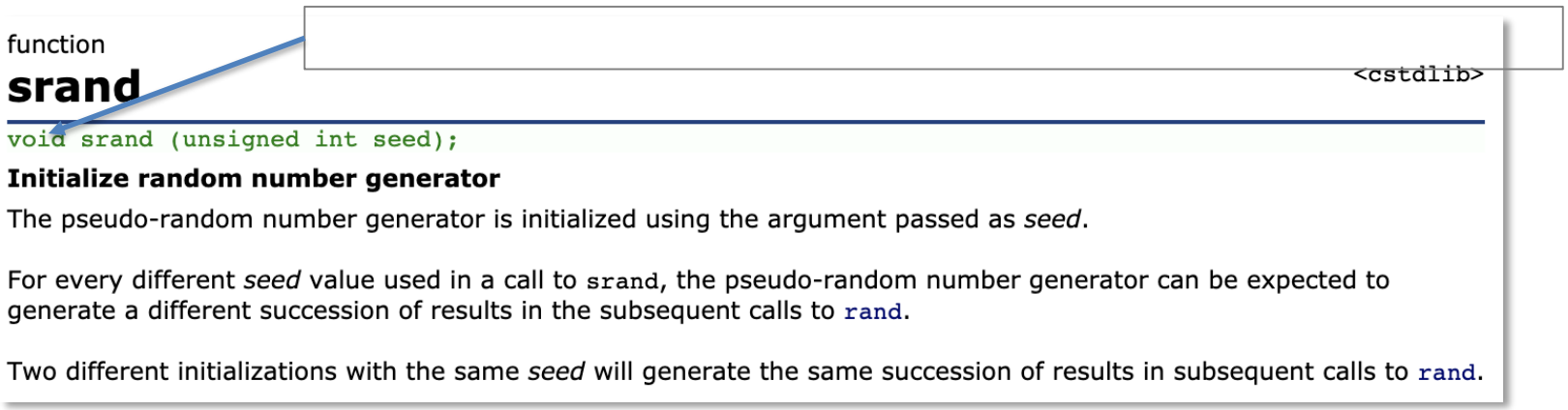
```
rand() % 101
```

```
rand() % 100 + 1
```

Example: Random Number Generation

```
#include <iostream>
#include <cstdlib>       // needed for calling rand()
using namespace std;

int main()
{
  for (int i = 0; i < 10; ++i)
    cout << rand() % 100 + 1 << endl;
  return 0;
}
```

Example: Random Number Generation



function

srand

<cstdlib>

```
void srand (unsigned int seed);
```

Initialize random number generator

The pseudo-random number generator is initialized using the argument passed as *seed*.

For every different *seed* value used in a call to `srand`, the pseudo-random number generator can be expected to generate a different succession of results in the subsequent calls to `rand`.

Two different initializations with the same *seed* will generate the same succession of results in subsequent calls to `rand`.

Example: Random Number Generation

```
#include <iostream>
#include <cstdlib>       // for calling srand(), rand()
using namespace std;

int main()
{
  srand(???);     // initialize the seed for rand()
  for (int i = 0; i < 10; ++i)
    cout << rand() % 100 + 1 << endl;
  return 0;
}
```

Example: Random Number Generation

function

time

time_t is a special data type for integral values representing time

<ctime>

```
time_t time (time_t* timer);
```

Get current time

Get the current calendar time as a value of type time_t

`time(NULL)`

```
// initialize random seed
srand(time(NULL));
```

Example: Random Number Generation

```
#include <iostream>
#include <cstdlib>      // for calling srand(), rand()
#include <ctime>        // for calling time()
using namespace std;

int main()
{
  srand(time(NULL)); // initialize the seed for rand()
  for (int i = 0; i < 10; ++i)
    cout << rand() % 100 + 1 << endl;
  return 0;
}
```

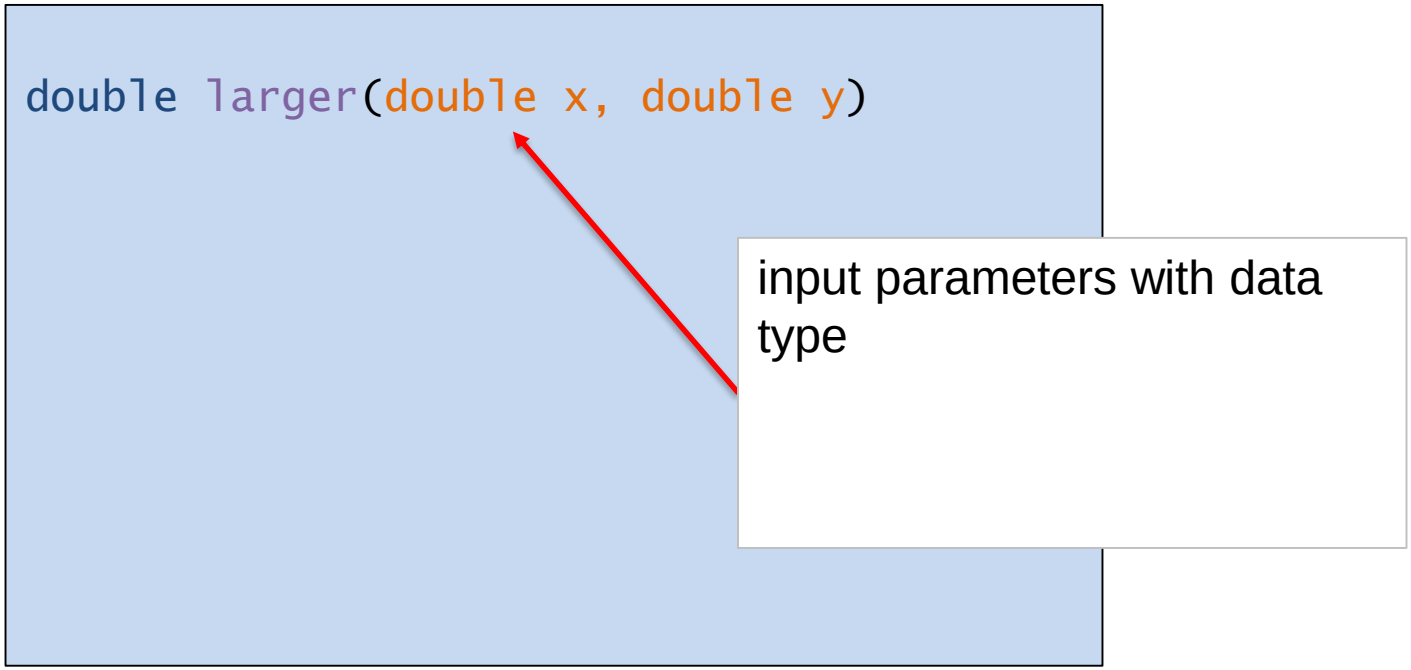
Example: Random Number Generation

```
srand(0)
```

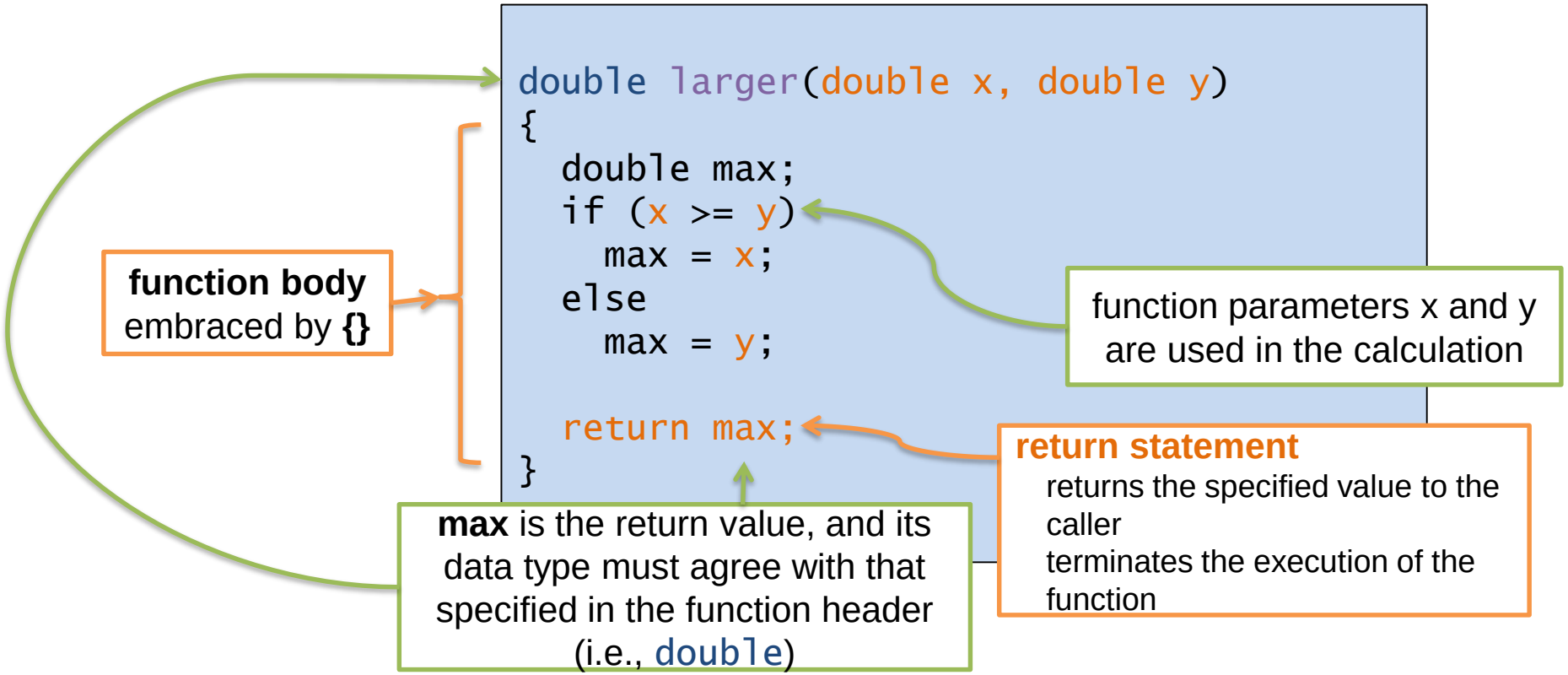
```
srand()
```

Defining Your Own Functions

Defining Your Own Functions

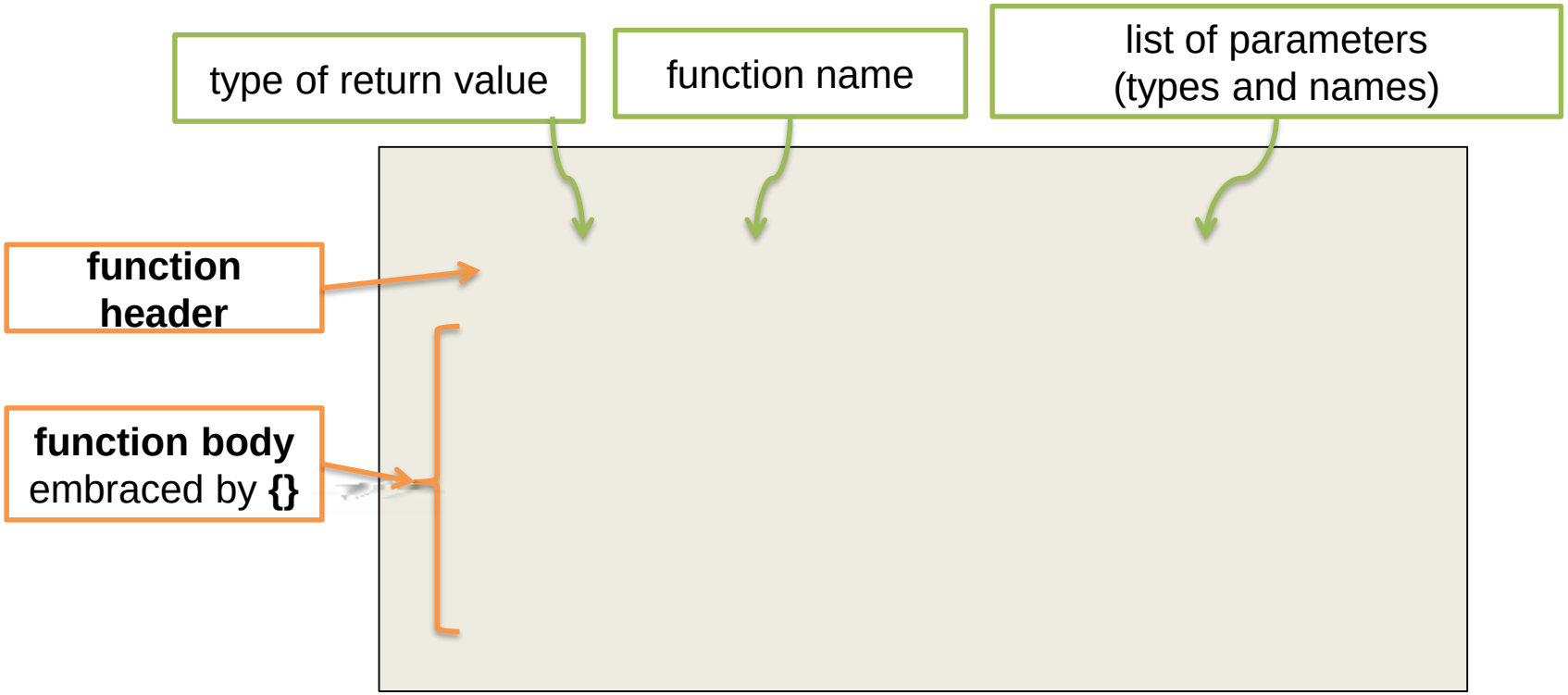


Defining Your Own Functions



FUNCTION DEFINITION, FUNCTION CALL &
FUNCTION DECLARATION

Function Definition

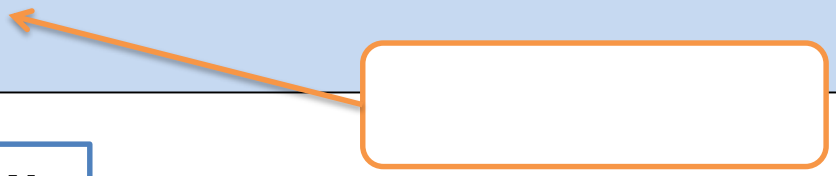


Void Functions

Void Functions

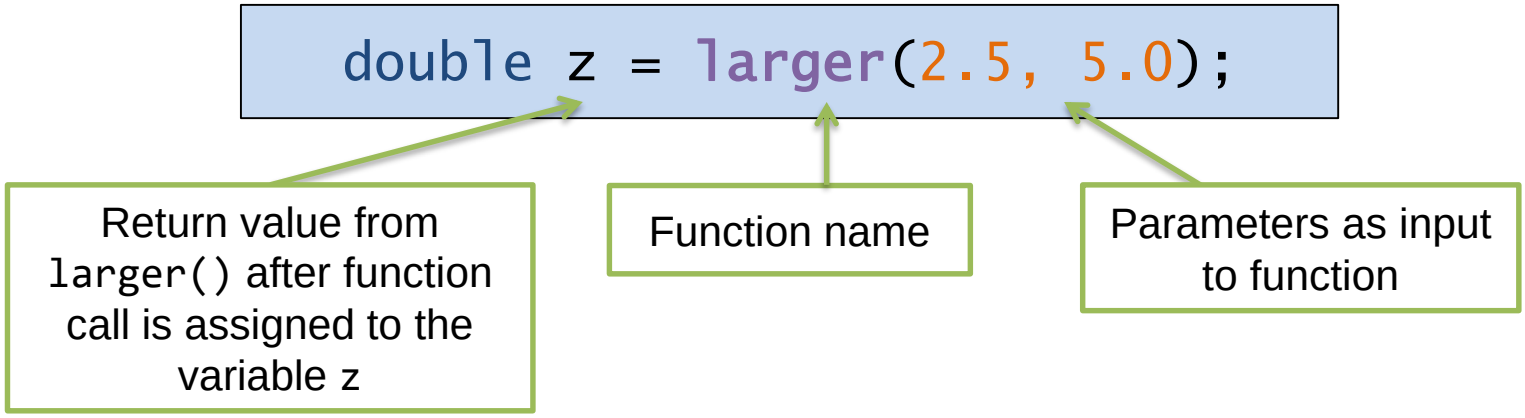
```
void print_msg(int x)
{
  cout << "This is a void function " << x << endl;
  return;
}
```

```
void print_msg(int x)
{
  cout << "This is a void function " << x << endl;
}
```



Both are OK!

Function Call



Function Call

```
double larger(double x, double y)
{
    …
}
```

parameters

```
c = larger ( 2.5, 5.0 );
```

arguments

Function Call

```
double z1 = larger (2.5, 5.0);                  // constants
double z2 = larger (one, two);                  // variables
double z3 = larger (one - 2, two);              // expressions
double z4 = larger (2.5, larger (3, 5.0) );  // a function
```

Function Declaration

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
    ….
    c = larger(a, b);
    ….
}
```

One way to do this is to place the function definition before the function call in the source file.

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

double larger(double x, double y);

int main()
{
  ….
  c = larger(a, b);
  ….
}

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}
```

Note the ; here. It is needed since this function declaration is a statement. Compare this with the function header in the example on the left.

Alternatively, the function definition can be placed anywhere in the source file by including a function declaration before the function call.

Function Declaration

Function Declaration

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

double larger(double p, double q) ;

int main()
{
  ….
  c= larger(a, b);
  ….
}

double larger(double x, double y)
{
  return (x >= y)? x : y;
}
```

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

double larger(double, double) ;

int main()
{
  ….
  c = larger(a, b);
  ….
}

double larger(double x, double y)
{
  return (x >= y)? x : y;
}
```

```
(x >= y)? x : y
```

FLOW OF CONTROL

Flow of Control

Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

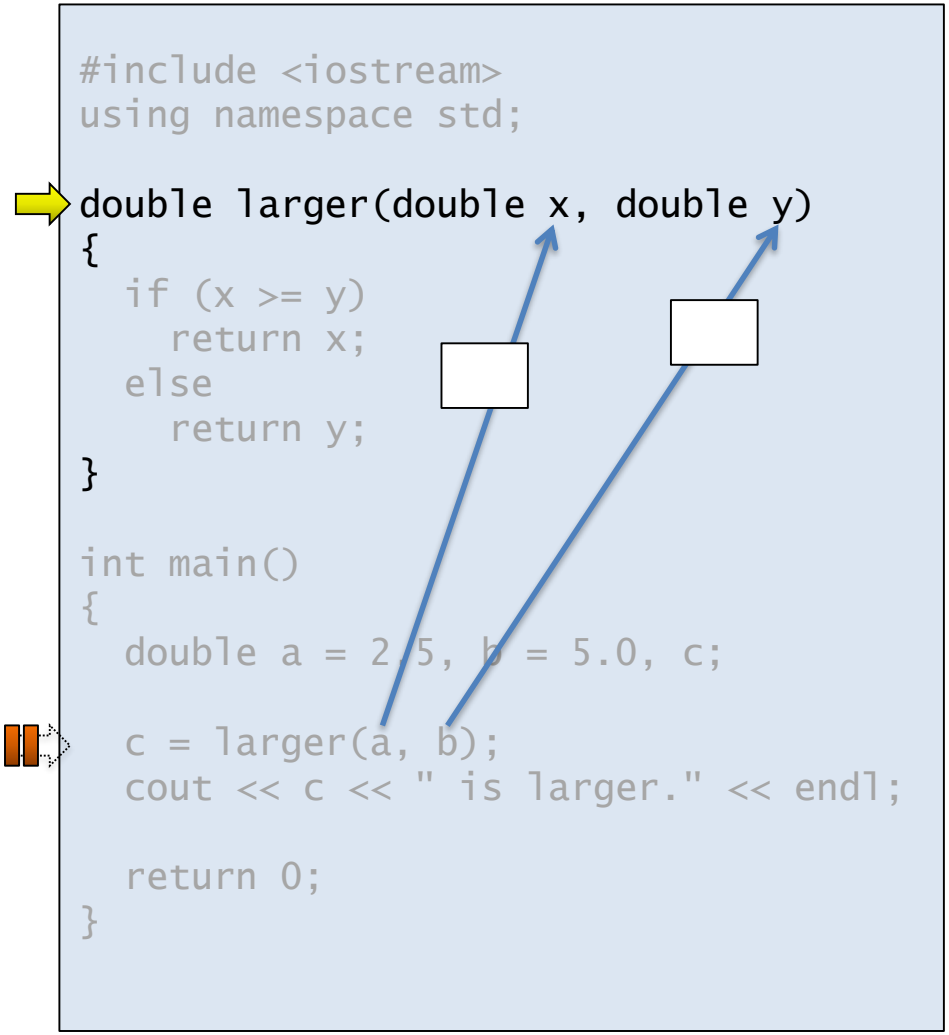
Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
   return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

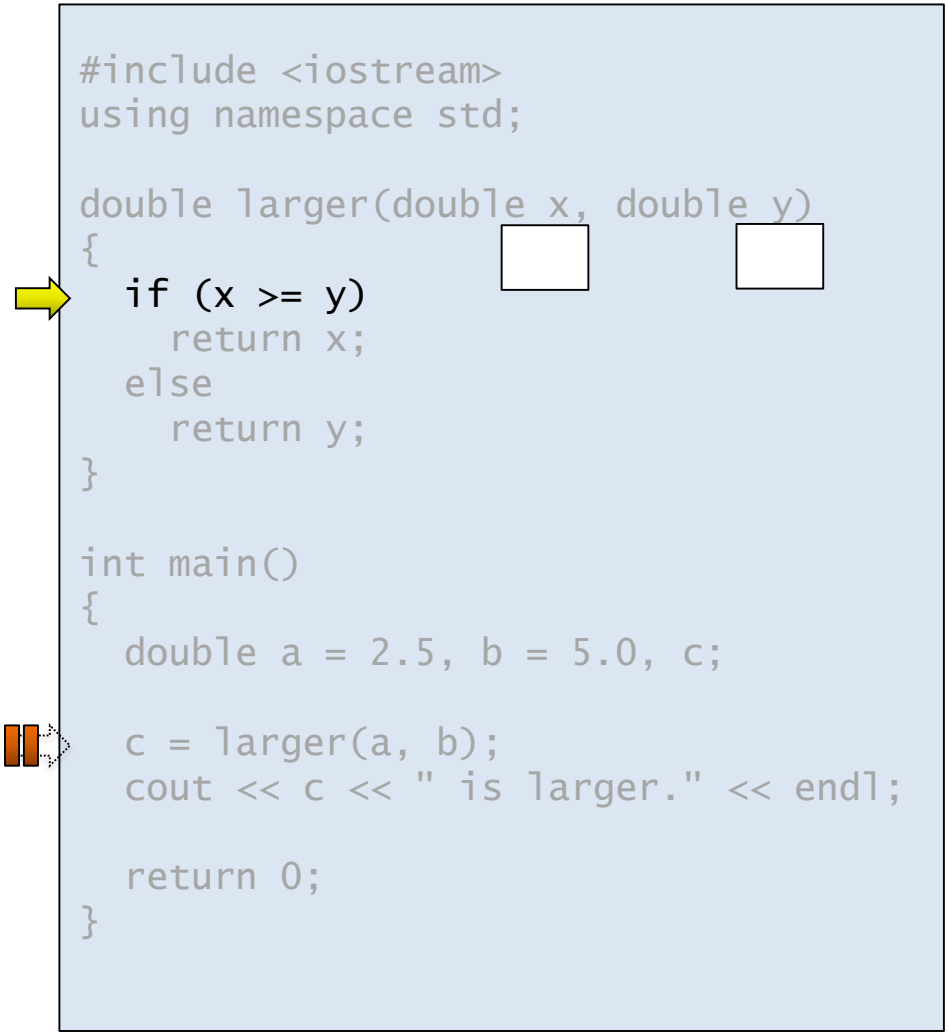
Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control

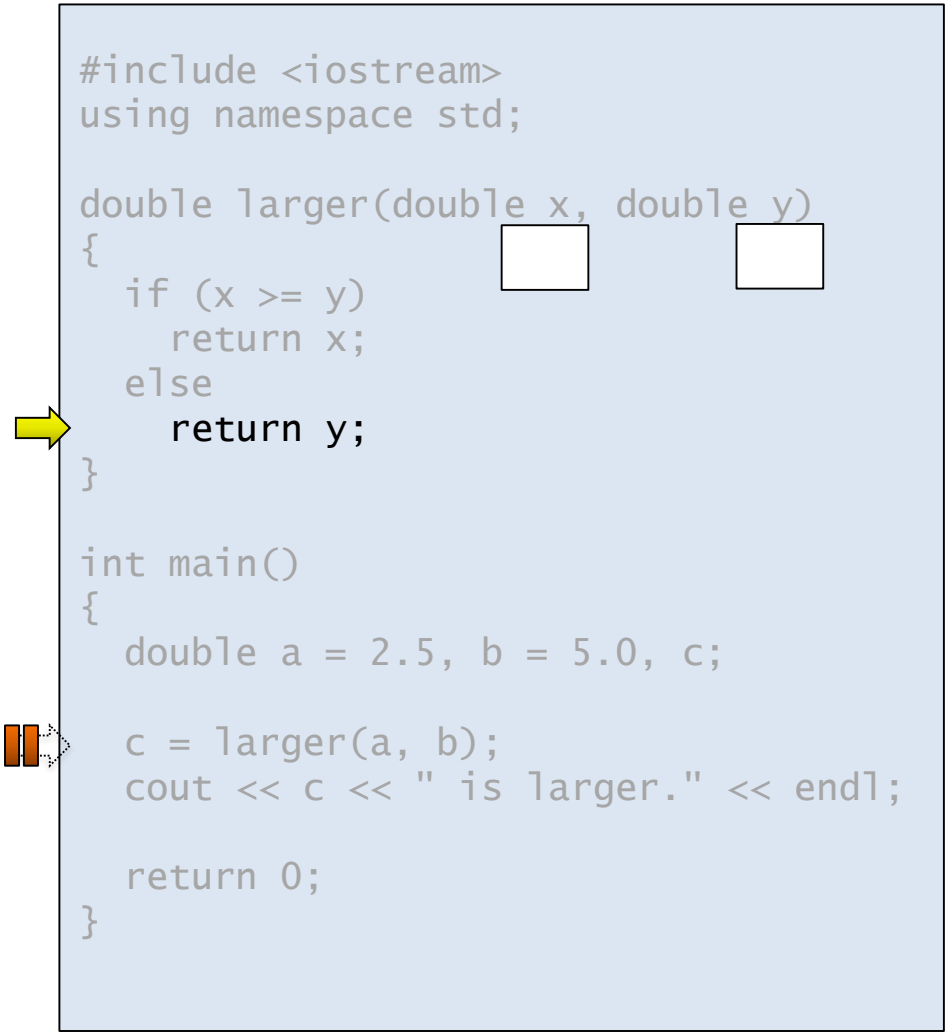


Flow of Control



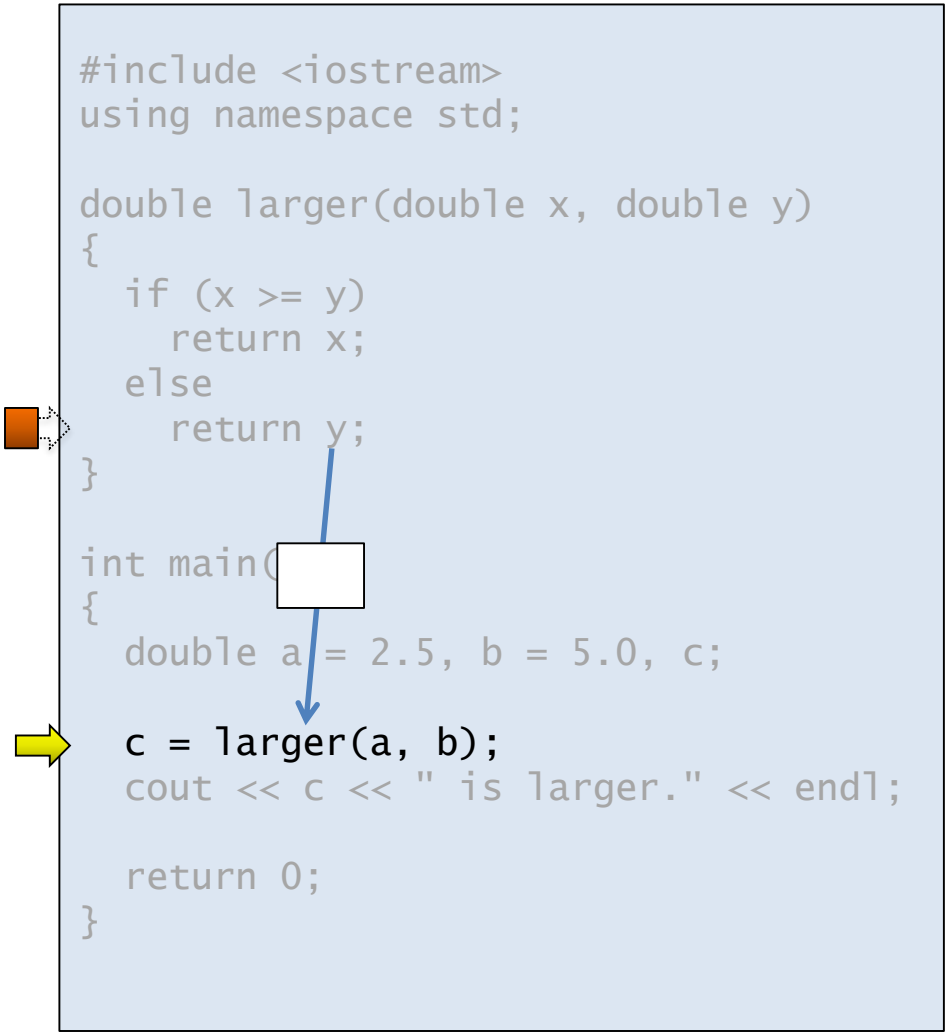
```
#include <iostream>
using namespace std;

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control



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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control



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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

c takes the value 5.0 which is the return value of `larger()`

Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

Flow of Control

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

double larger(double x, double y)
{
  if (x >= y)
    return x;
  else
    return y;
}

int main()
{
  double a = 2.5, b = 5.0, c;

  c = larger(a, b);
  cout << c << " is larger." << endl;

  return 0;
}
```

SCOPE OF VARIABLES

Local Variables

Local Variables

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

double larger(double x, double y)
{
    double ym;
    max = (x >= y)? x : y;

    return max;
}

int main()
{
    double m = 2.5, n = 5.0, ym;

    max = larger(a, b);
    cout << max << " is larger." << endl;

    return 0;
}
```

local variables of **larger()**:
x, y, max
i.e., these variables are input parameters or variables defined in the function larger(), and therefore can only be seen or used in larger()

local variables of **main()**:
a, b, max
i.e., these variables are defined in the function main(), and therefore can only be seen or used in main()

The local variables **max** of **larger()** and **max** of **main()** are **unrelated**.

Local Variables

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

double larger(double x, double y)
{
          nxq ym
   max = (x >= y)? x : y;

   return max;
}

int main()
{
   double a = 2.5, b = 5.0, max;

   max = larger(a, b);
   cout << max << " is larger." << endl;

   return 0;
}
```

There will be a compilation error if we comment out the declaration of **max** in **larger()** because **max** in **main()** is a local variable of **main()** and cannot be seen or used in larger().

Global Variables

Global Variables

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

double m, n;
const double PI = 3.1415;

double larger()
{
   return (a >= b)? a : b;
}

int main()
{
   cout << "Input two integers: ";
   cin >> a >> b;
   cout << larger() << " is larger." << endl;

   double r;
   cout << "Input radius of a circle: ";
   cin >> r;
   cout << "Area of circle = " << PI * r * r << endl;
   return 0;
}
```

global variables:
a, b, PI

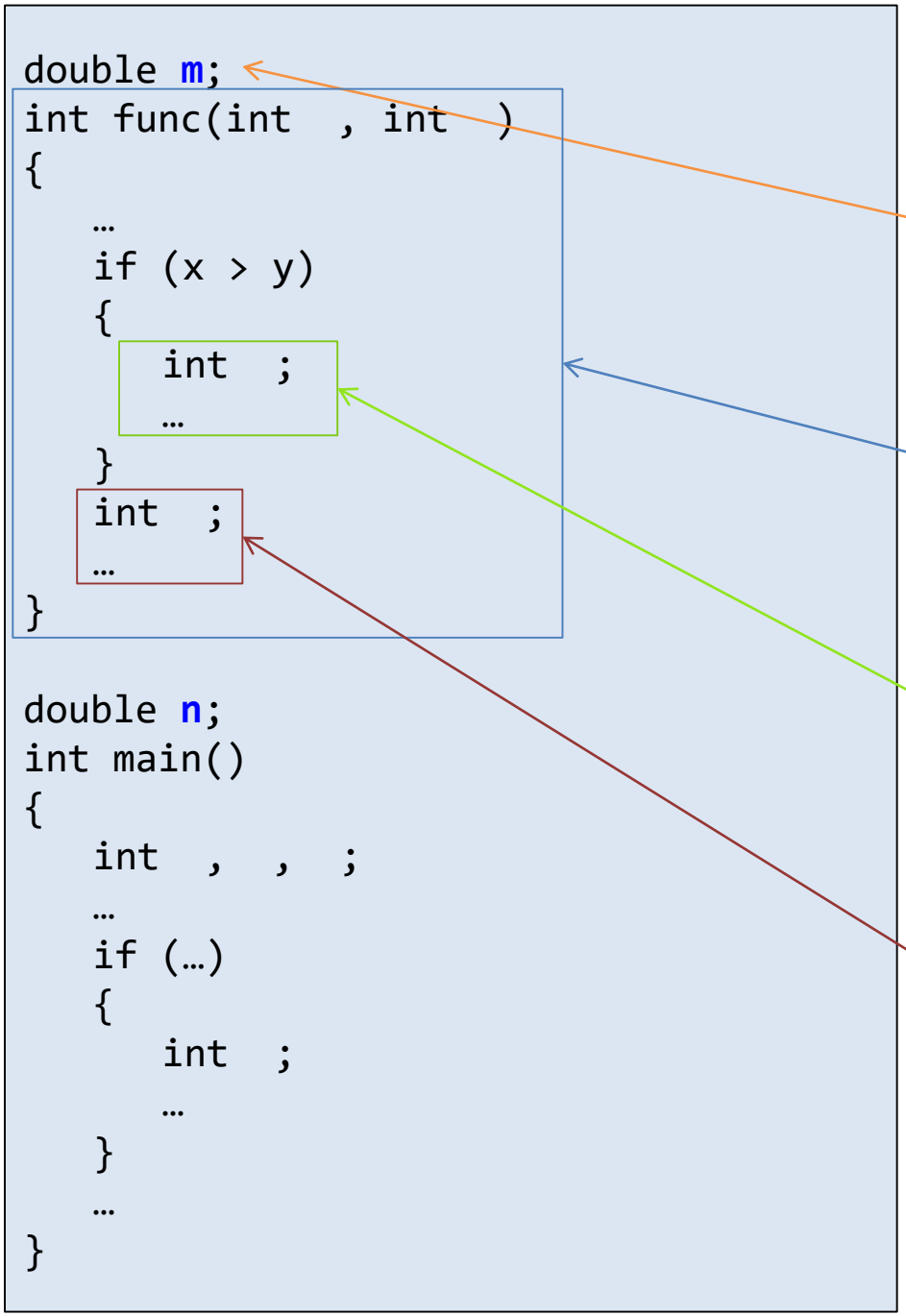
The global constant **PI** can be used throughout the file after its declaration.

Avoid using global variables to communicate data between functions

The variables **a, b** should best be changed into input parameters for the function larger(). Can you do that?

Scopes of Variables

Scopes of Variables



Scope of global variable **a**:
from declaration to end of block
(in this case, end of file; hence scope of **a** is the entire file)

Scope of formal parameters **x, y**:
entire function

Scope of local variable **k**:
from declaration to end of block
(in this case, end of if statement)

Scope of local variable **z**:
from declaration to end of block
(in this case, end of func)

Scopes of Variables

```
double m;
int func(int  , int  )
{
   …
   if (x > y)
   {
      int  ;
      …
   }
   int  ;
   …
}

double n;
int main()
{
   int  ,  ,  ;
   …
   if (…)
   {
      int  ;
      …
   }
   …
}
```

Scope of global variable **b**:
from declaration to end of block
(in this case, end of file)

Scope of local variables **x, y, z**:
from declaration to end of block
(in this case, end of main function)

Scope of local variable **x** in the inner block:
from declaration to end of block
(in this case, end of if statement)

the outer x is hidden within this block

Scopes of Variables

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

int main()
{
   int i = 0;
   cout << "Outer block: i = " << i << endl;

   {
      int i = 100;
      cout << "Inner block: i = " << i << endl;
   }

   cout << "Outer block: i = " << i << endl;
   return 0;
}
```

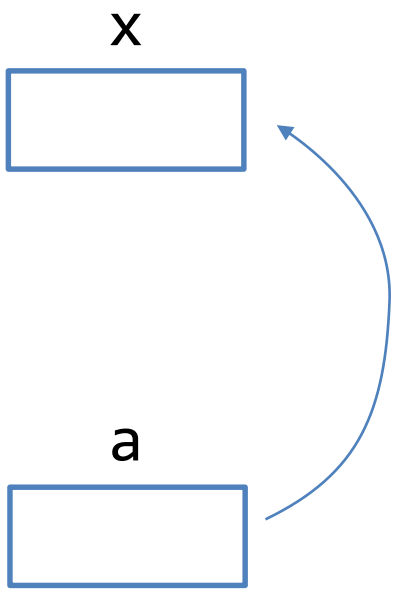
```
Outer block: i = 0
Inner block: i = 100
Outer block: i = 0
```

PARAMETER PASSING MECHANISM

Pass-by-Value

Pass-by-Value

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

// computes the square of an integer
void square( int x )
{
   x *= x;
}

int main()
{
   int a = 10;
   cout << a << " squared: ";
   square( a );
   cout << a << endl;
   return 0; }
```

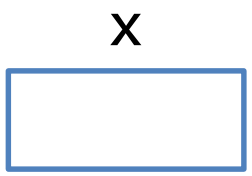


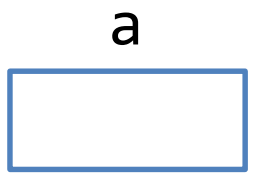
x is a parameter and also a local variable of the square function

Copying of value of actual argument to formal parameter

Pass-by-Value

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

// computes the square of an integer
void square( int x )
{
   x *= x;
}

int main()
{
   int a = 10;
   cout << a << " squared: ";
   square( a );
   cout << a << endl;
   return 0; }
```

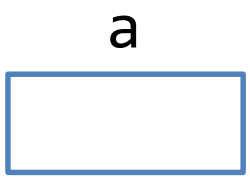




Pass-by-Value

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

// computes the square of an integer
void square( int x )
{
   x *= x;
}

int main()
{
   int a = 10;
   cout << a << " squared: ";
   square( a );
   cout << a << endl;
   return 0; }
```

Variable **x** disappears (more precisely, the memory location it occupies is released back to the system) upon function completion.



Pass-by-Value

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

void swap(int a, int b)
{
    cout << "a = " << a << ", b = " << b << endl;
    int temp = a;
    a = b;
    b = temp;
    cout << "a = " << a << ", b = " << b << endl;
}

int main()
{
    int x = 0, y = 100;
    cout << "x = " << x << ", y = " << y << endl;
    swap(x, y);
    cout << "x = " << x << ", y = " << y << endl;
    return 0;
}
```

```
x = 0, y = 100
a = 0, b = 100
a = 100, b = 0
x = 0, y = 100
```

Because the variables x and y are passed to swap() using pass-by-value, only the values are transferred to swap(), and swap() can only deal with its local variables a and b.

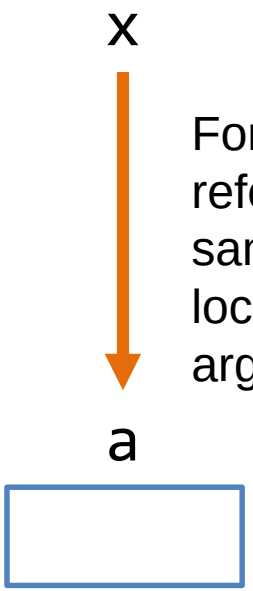
Pass-by-Reference

Pass-by-Reference

Pass-by-Reference

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

// computes the square of an integer
void square( int &x )
{
   x *= x;
}

int main()
{
   int a = 10;
   cout << a << " squared: ";
   square( a );
   cout << a << endl;
   return 0;
}
```

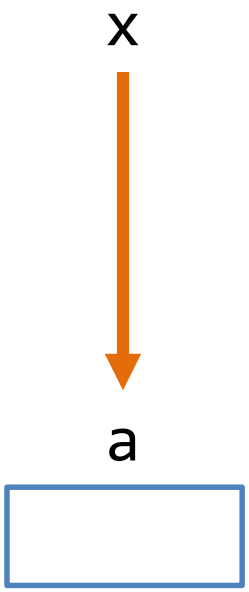
Note the **&** to indicate that the formal parameter x is pass-by-reference.



Formal parameter refers to the same memory location as the argument

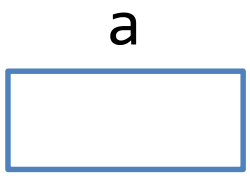
Pass-by-Reference

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

// computes the square of an integer
void square( int &x )
{
   x *= x;
}

int main()
{
   int a = 10;
   cout << a << " squared: ";
   square( a );
   cout << a << endl;
   return 0;
}
```



Pass-by-Reference

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

// computes the square of an integer
void square( int &x )
{
   x *= x;
}

int main()
{
   int a = 10;
   cout << a << " squared: ";
   square( a );
   cout << a << endl;
   return 0;
}
```



Pass-by-Reference

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

void swap(int &a, int &b)
{
    cout << "a = " << a << ", b = " << b << endl;
    int temp = a;
    a = b;
    b = temp;
    cout << "a = " << a << ", b = " << b << endl;
}

int main()
{
    int x = 0, y = 100;
    cout << "x = " << x << ", y = " << y << endl;
    swap(x, y);
    cout << "x = " << x << ", y = " << y << endl;
    return 0;
}
```

```
x = 0, y = 100
a = 0, b = 100
a = 100, b = 0
x = 100, y = 0
```

The formal parameters a and b in swap() refer to the memory locations of the arguments x and y, respectively.

Pass-by-Reference vs. Value-Returning Function

```
uz  squareByValue( uz  z ynq  )
{
      return number *= number;
}
```

```
  u  squareByReference( uz   z ynq  )
{
     number *= number;
}
```

Pass-by-Reference vs. Value-Returning Function

```
int squareByValue( int );
void squareByReference( int & );

int main()
{
     int x = 2;
     int z = 4;

     cout << "x = " << x << " before squareByValue\n";
     cout << "Value returned by squareByValue: "
          <<    m q   mx q      << endl;
     cout << "x = " << x << " after squareByValue\n" << endl;

     cout << "z = " << z << " before squareByReference" << endl;
        m q   qrq qzoq
     cout << "z = " << z << " after squareByReference" << endl;

     return 0;
}
```

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

Return value of squareByValue() is used by the cout expresssion.

Result of computation by squareByReference() is updated in z.

Pass-by-Reference vs. Value-Returning Function

```
const double CONVERSION = 2.54;
const int INCHES_IN_FOOT = 12;
const int CENTIMETERS_IN_METER = 100;

  u  metersAndCentTofeetAndInches(int mt, int ct, uz   r, uz   uz)
{
     int centimeters;
     centimeters = mt * CENTIMETERS_IN_METER + ct;
     in = (int) (centimeters / CONVERSION);
     r = in / INCHES_IN_FOOT;
     uz = in % INCHES_IN_FOOT;
}
```

f and in are the computation results.
Think about how the calling functions can call this function and access the results through the arguments after function call.

Quick Exercise 1

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

void figureMeOut(int &x, int y, int &z) {
    cout << x << ' ' << y << ' ' << z << endl;
    x = 1;
    y = 2;
    z = 3;
    cout << x << ' ' << y << ' ' << z << endl;
}

int main() {
    int a=10, b=20, c=30;
    figureMeOut(a, b, c);
    cout << a << ' ' << b << ' ' << c << endl;
}
```

Answer to Quick Exercise 1

```
10 20 30
1 2 3
1 20 3
```

We are happy to help you!

“If you face any problems in understanding the materials,

.

We wish you enjoy learning programming in this class ☺.”