Lecture 9

Scope & Recursion

UBS
ITU INFORMATION TECHNOLOGY UNIVERSITY

QUIZ

سورة طه پارہ: 16

قَالَ رَبِّ اشْرَحْ لِيْ صَدْرِيْ ۙ ﴿٢٥﴾

[قَالَ: اس نے کہا] [رَبِّ: اے رب !] [اشْرَحْ: کھول دے] [لِيْ: میرے لیے] [صَدْرِيْ : میرا سینہ]

وَيَسِّرْ لِيْٓ اَمْرِيْ ۙ ﴿٢٦﴾

[وَيَسِّرْ: اور آسان کر دے] [لِيْ: میرے لیے] [اَمْرِيْ : میرا کام]

وَاحْلُلْ عُقْدَةً مِّنْ لِّسَانِيْ ۙ ﴿٢٧﴾

[وَاحْلُلْ: اور کھول دے] [عُقْدَةً: گرہ] [مِّنْ: سے] [لِّسَانِيْ: میری زبان]

يَفْقَهُوْا قَوْلِيْ ۖ ﴿٢٨﴾

[يَفْقَهُوْا: وہ سمجھ سکیں] [قَوْلِيْ : میری بات]

QUESTIONS / FEEDBACK / CONCERNS

UBS

ITU INFORMATION TECHNOLOGY UNIVERSITY

SE SEC A
SLIDE OF FAME

NO ONE

WEEK-1

Muhammad Daniyal Hammad (BSSE23046)

WEEK-2

YOUR NAME

WEEK-3

YOUR NAME

WEEK-4

YOUR NAME

WEEK-5

YOUR NAME

WEEK-6

YOUR NAME

WEEK-7

YOUR NAME

WEEK-8

YOUR NAME

WEEK-9

YOUR NAME

WEEK-10

YOUR NAME

MIDTERM

YOUR NAME

WEEK-11

YOUR NAME

WEEK-12

YOUR NAME

WEEK-13

YOUR NAME

WEEK-14

YOUR NAME

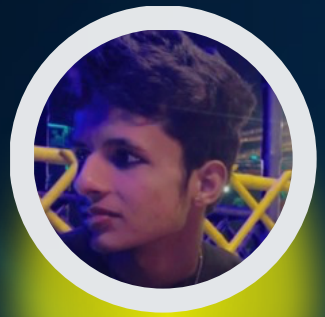
WEEK-15

UBS

ITU INFORMATION TECHNOLOGY UNIVERSITY

SE SEC B
SLIDE OF FAME

Muhammad Mukarram
BSSE23029
WEEK-1



Muhammad Abdullah
(BSSE23087)
WEEK-2

YOUR NAME
WEEK-3

YOUR NAME
WEEK-4

YOUR NAME
WEEK-5

YOUR NAME
WEEK-6

YOUR NAME
WEEK-7

YOUR NAME
WEEK-8

YOUR NAME
WEEK-9

YOUR NAME
WEEK-10

YOUR NAME
MIDTERM

YOUR NAME
WEEK-11

YOUR NAME
WEEK-12

YOUR NAME
WEEK-13

YOUR NAME
WEEK-14

YOUR NAME
WEEK-15

UBS

ITU
INFORMATION
TECHNOLOGY
UNIVERSITY

What I can control and what I can't

Data source: @mindfulenough | Infographic design by @agrassoblog for educational and motivational purposes

INFORMATION TECHNOLOGY UNIVERSITY

RECAP

GitHub

Tools (Cygwin, IDE, GitHub)

Flowcharts

Algorithms

Approach towards a word problem

Pseudocode

Flowcharts Advantages & Disadvantages

Numbers Systems (Decimal, Binary, Octal & Hexadecimal)

Ten’s Complement

Twos Complement

main function

Stream in and stream out operators

if else

Functions

Data Types

Arithmetic Operators

Relational Operators

Loops (While, for , do while)

Switch cases

Function Overloading

RECAP

Scope of variables

Function Prototype and Definition

- Function declarations need to occur before invocations

```
int foo()
{
    return bar()*2; // ERROR - bar hasn't been declared yet
}

int bar()
{
    return 3;
}
```

- Function declarations need to occur before invocations
  - Solution 1: reorder function declarations

```
int bar()
{
    return 3;
}

int foo()
{
    return bar()*2; // ok
}
```

- Function declarations need to occur before invocations
  - Solution 1: reorder function declarations
  - Solution 2: use a function prototype; informs the compiler you'll implement it later

```
int bar();

int foo()
{
    return bar()*2; // ok
}

int bar()
{
    return 3;
}
```

- Function prototypes should match the signature of the method, though argument names don't matter

```
int square(int);

int cube(int x)
{
    return x*square(x);
}

int square(int x)
{
    return x*x;
}
```

- Function prototypes should match the signature of the method, though argument names don't matter

```
int square(int x);

int cube(int x)
{
    return x*square(x);
}

int square(int x)
{
    return x*x;
}
```

- Function prototypes should match the signature of the method, though argument names don't matter

```
int square(int z);

int cube(int x)
{
    return x*square(x);
}

int square(int x)
{
    return x*x;
}
```

- Function prototypes are generally put into separate header files
  - Separates specification of the function from its implementation

```
// myLib.h - header
// contains prototypes

int square(int);
int cube (int);
```

```
// myLib.cpp - implementation
#include "myLib.h"

int cube(int x)
{
    return x*square(x);
}

int square(int x)
{
    return x*x;
}
```

Functions

```
// Function capital() with two default arguments
// Prototype:
double capital( double k0, double p=3.5, double n=1.0);

double endcap;

endcap = capital( 100.0, 3.5, 2.5);
endcap = capital( 2222.20, 4.8);
endcap = capital( 3030.00);

endcap = capital( );


endcap = capital( 100.0, , 3.0);


endcap = capital( , 5.0);
```

Image Reference : A Complete Guide to Programming in C++

Functions

```
// Function capital() with two default arguments
// Prototype:
double capital( double k0, double p=3.5, double n=1.0);

double endcap;

endcap = capital( 100.0, 3.5, 2.5);   // ok
endcap = capital( 2222.20, 4.8);      // ok
endcap = capital( 3030.00);           // ok

endcap = capital( );                  // not ok
// The first argument has no default value.

endcap = capital( 100.0, , 3.0);      // not ok
// No gap!

endcap = capital( , 5.0);             // not ok
// No gap either.
```

Global Variables

- How many times is function foo() called? Use a global variable to determine this.
  - Can be accessed from any function

```
int numCalls = 0;

void foo() {
  ++numCalls;
}

int main() {
  foo(); foo(); foo();
  cout << numCalls << endl;
}
```

Scope

- Scope: where a variable was declared, determines where it can be accessed from

```
int numCalls = 0;
int raiseToPower(int base, int exponent) {
        numCalls = numCalls + 1;
        int result = 1;
        for(int i = 0; i < exponent; i = i +1)
        {
            result= result * base;
        }
        return result;
}

int max(int num1, int num2)
{ numCalls = numCalls + 1;
  int result;
  if(num1> num2)
     result = num1;
  else
     result = num2;
  return result;
}
```

Scope

- Scope: where a variable was declared, determines where it can be accessed from
- numCalls has global scope – can be accessed from any function

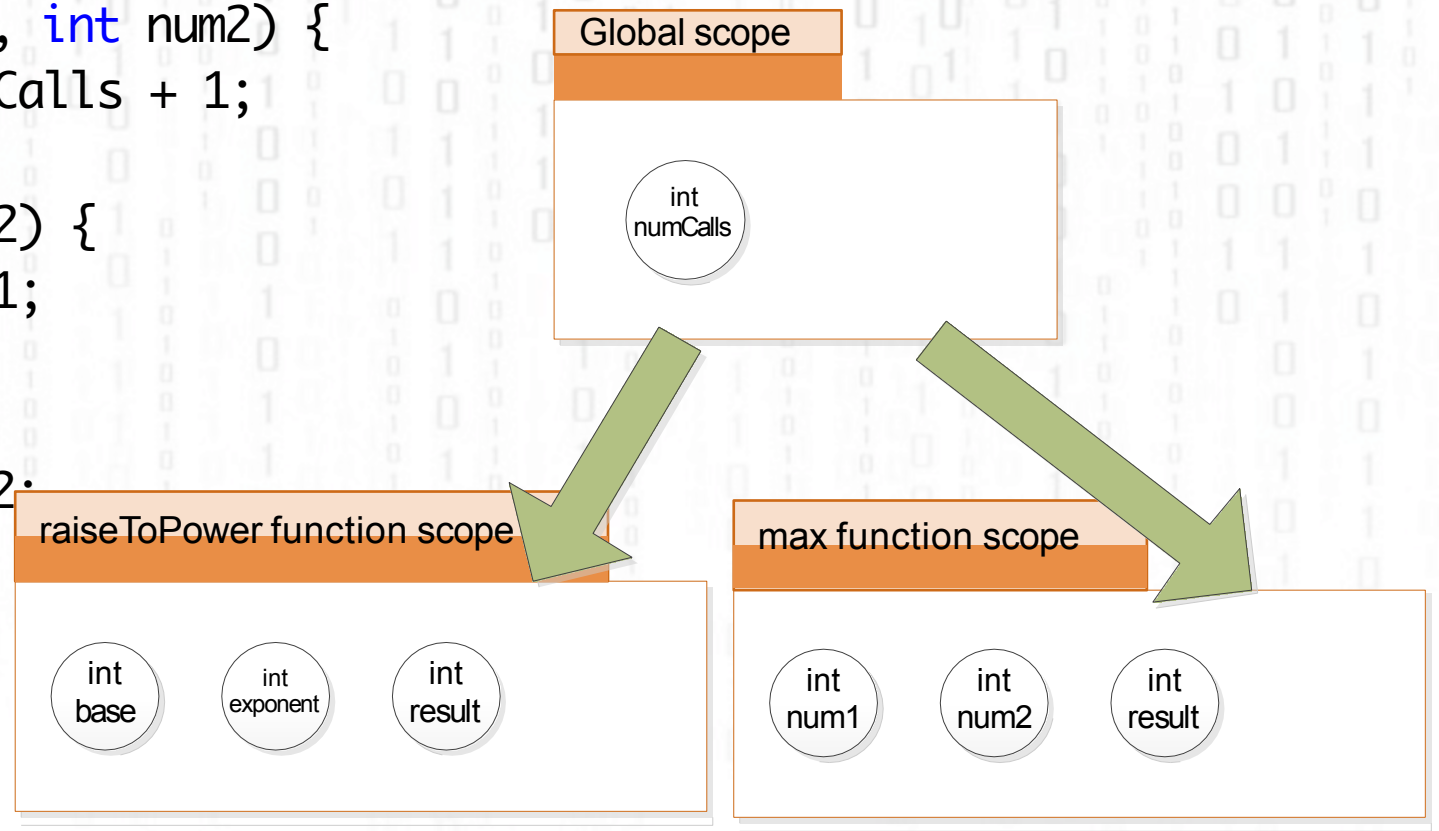
```
int numCalls = 0;
int raiseToPower(int base, int exponent) {
        numCalls = numCalls + 1;
        int result = 1;
        for(int i = 0; i < exponent; i = i +1)
        {
            result= result * base;
        }
        return result;
}

int max(int num1, int num2)
{ numCalls = numCalls + 1;
  int result;
  if(num1> num2)
     result = num1;
  else
     result = num2;
  return result;
}
```

Scope

- Scope: where a variable was declared, determines where it can be accessed from
  - numCalls has global scope – can be accessed from any function
  - result has function scope – each function can have its own separate variable named result

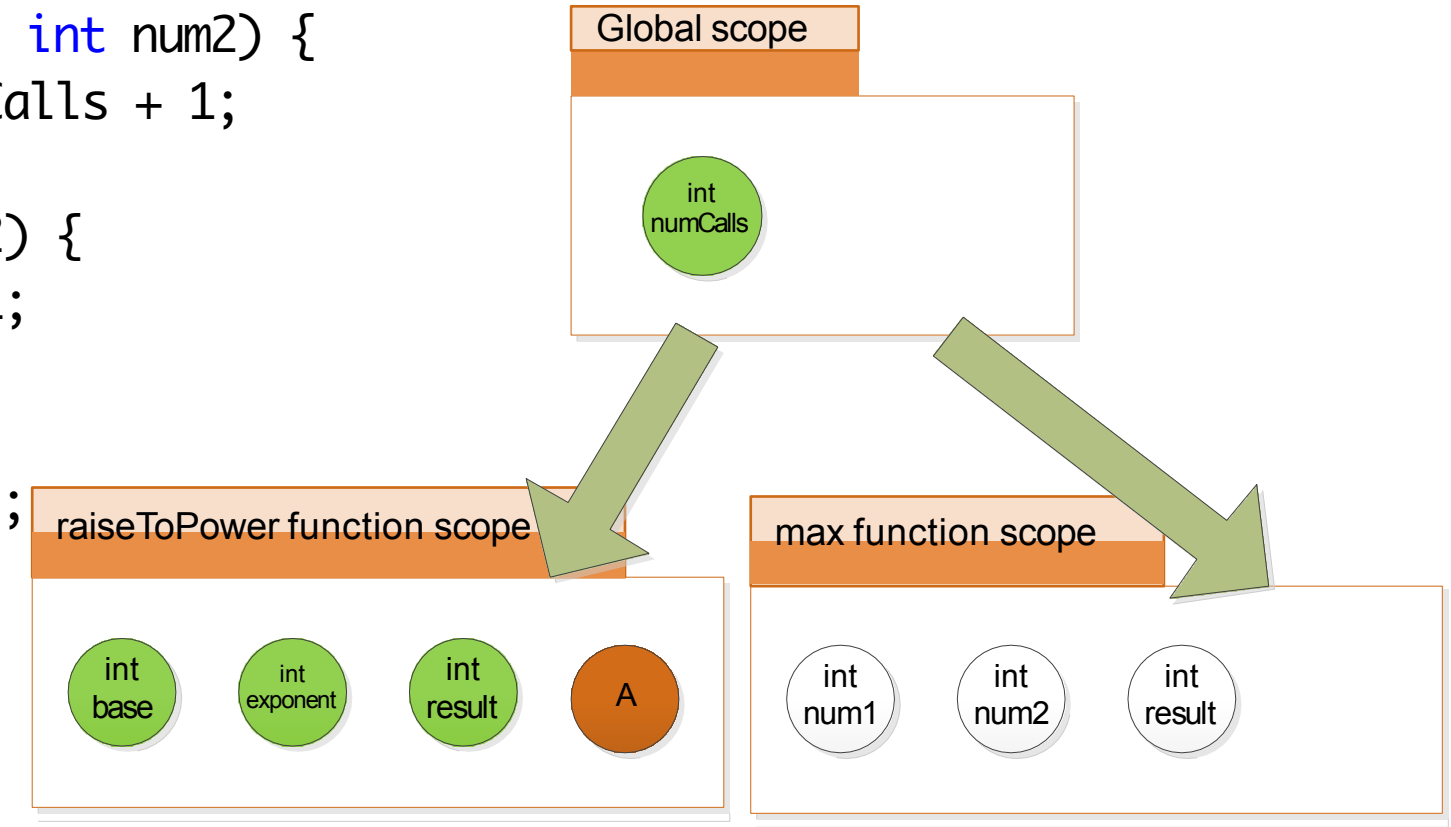
```
int numCalls = 0;
int raiseToPower(int base, int exponent) {
        numCalls = numCalls + 1;
        int result = 1;
        for(int i = 0; i < exponent; i = i +1)
        {
            result= result * base;
        }
        return result;
}

int max(int num1, int num2)
{ numCalls = numCalls + 1;
  int result;
  if(num1> num2)
     result = num1;
  else
     result = num2;
  return result;
}
```

```
int numCalls = 0;
int raiseToPower(int base, int exponent)
  { numCalls = numCalls + 1;
  int result = 1;
  for (int i = 0; i < exponent; i = i + 1) {
    result = result * base;
  }
  // A
  return result;
}int max(int num1, int num2) {
   numCalls = numCalls + 1;
   int result;
   if (num1 > num2) {
     result = num1;
   }
   else {
     result = num2;
   }
   // B
   return result;
}
```



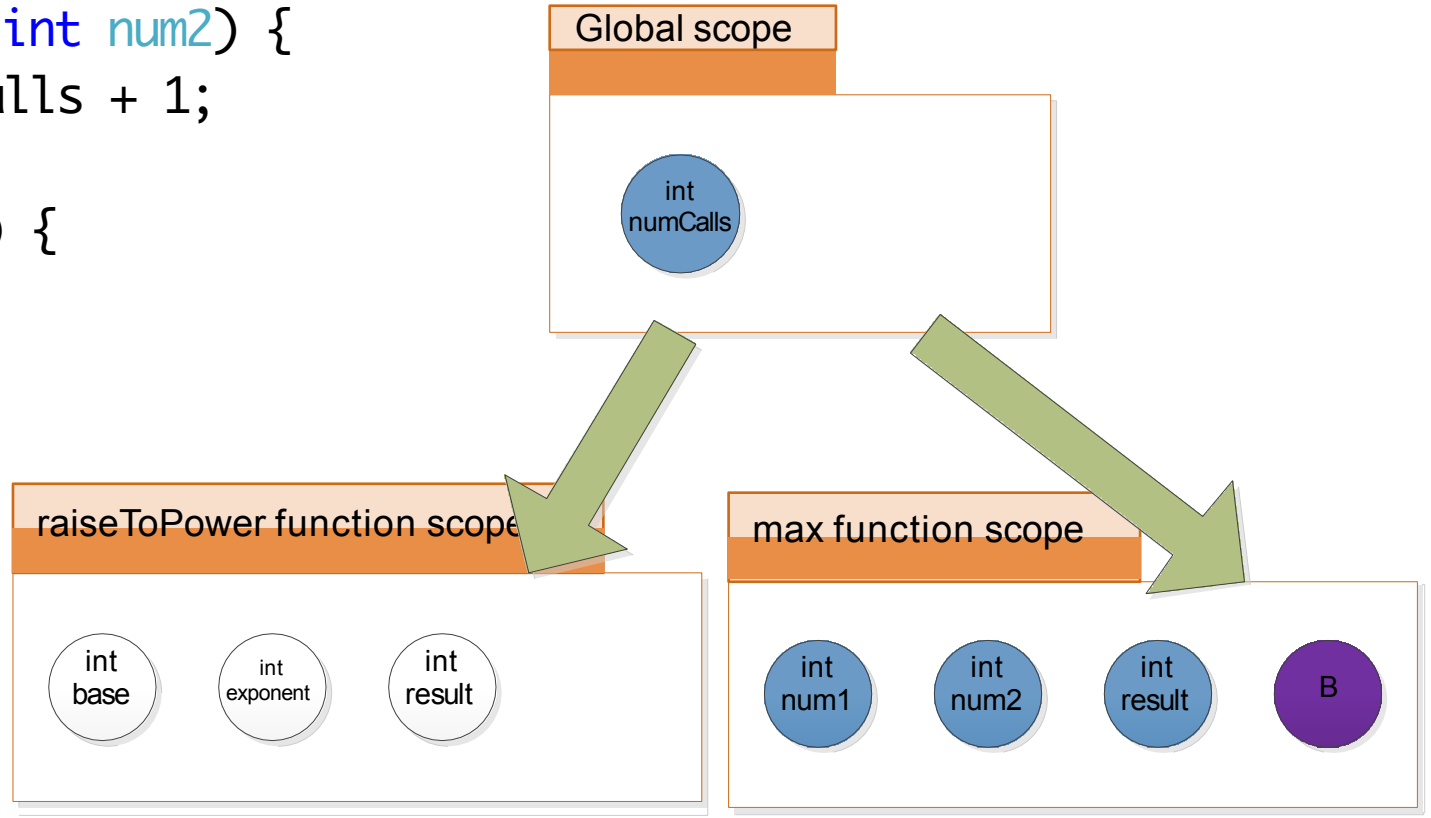
```
int numCalls = 0;
int raiseToPower(int base, int exponent) {
  numCalls = numCalls + 1;
  int result = 1;
  for (int i = 0; i < exponent; i = i + 1) {
    result = result * base;
  }// A
  return result;
}
int max(int num1, int num2) {
  numCalls = numCalls + 1;
  int result;
  if (num1 > num2) {
    result = num1;
  }
  else {
    result = num2;
  }
  // B
  return result;
}
```

- At A, variables marked in green are in scope



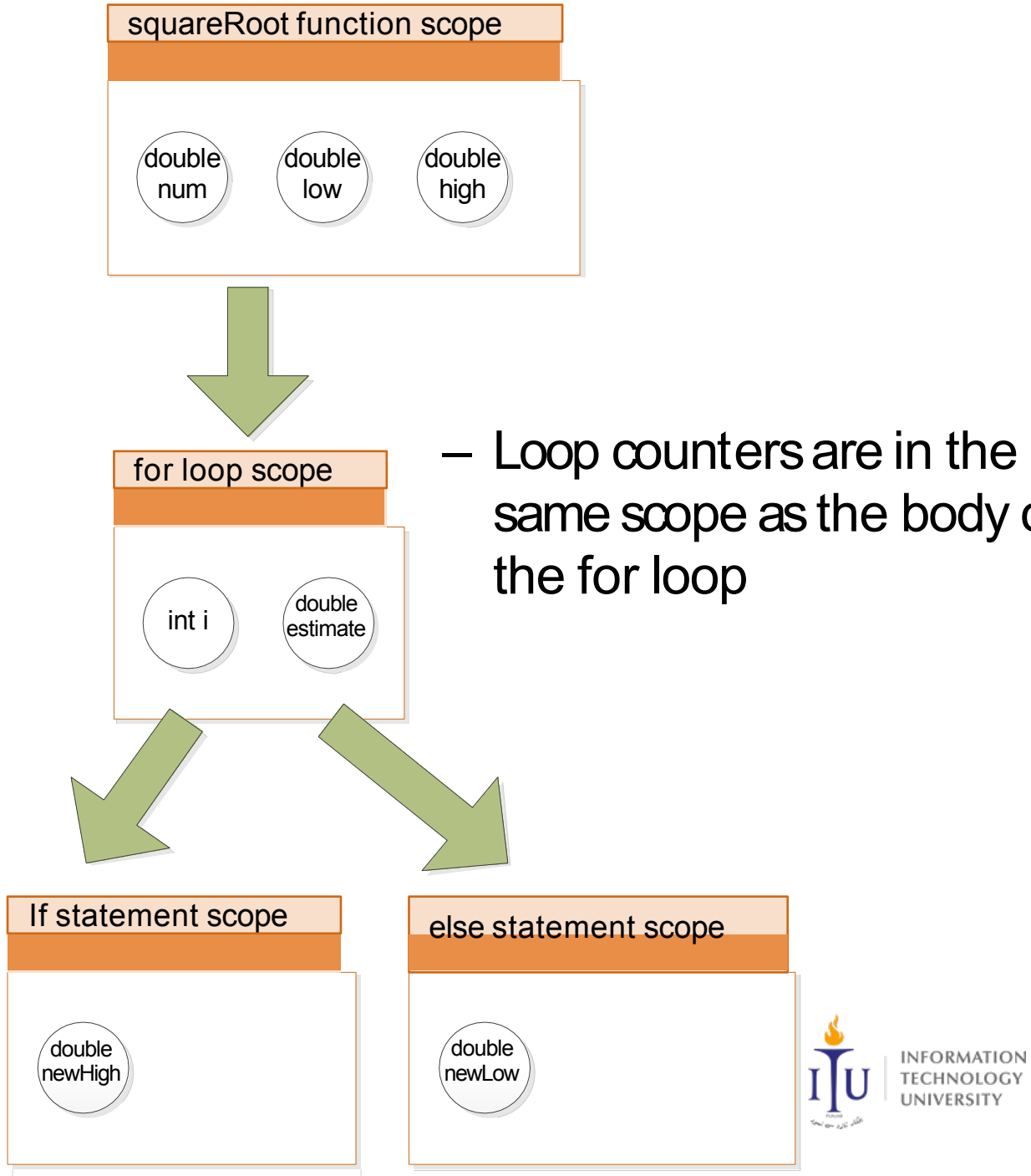
```
int numCalls = 0;
int raiseToPower(int base, int exponent) {
  numCalls = numCalls + 1;
  int result = 1;
  for (int i = 0; i < exponent; i = i + 1) {
    result = result * base;
  }// A
  return result;
}
int max(int num1, int num2) {
  numCalls = numCalls + 1;
  int result;
  if (num1 > num2) {
    result = num1;
  }
  else {
    result = num2;
  }
  // B
  return result;
}
```

- At B, variables marked in blue are in scope



```
double squareRoot(double num) {
  double low = 1.0;
  double high = num;
  for (int i = 0; i < 30; i = i + 1) {
    double estimate = (high + low) / 2;
    if (estimate*estimate > num) {
      double newHigh = estimate;
      high = newHigh;
    } else {
      double newLow = estimate;
      low = newLow;
    }
  }
  return (high + low) / 2;
}
```

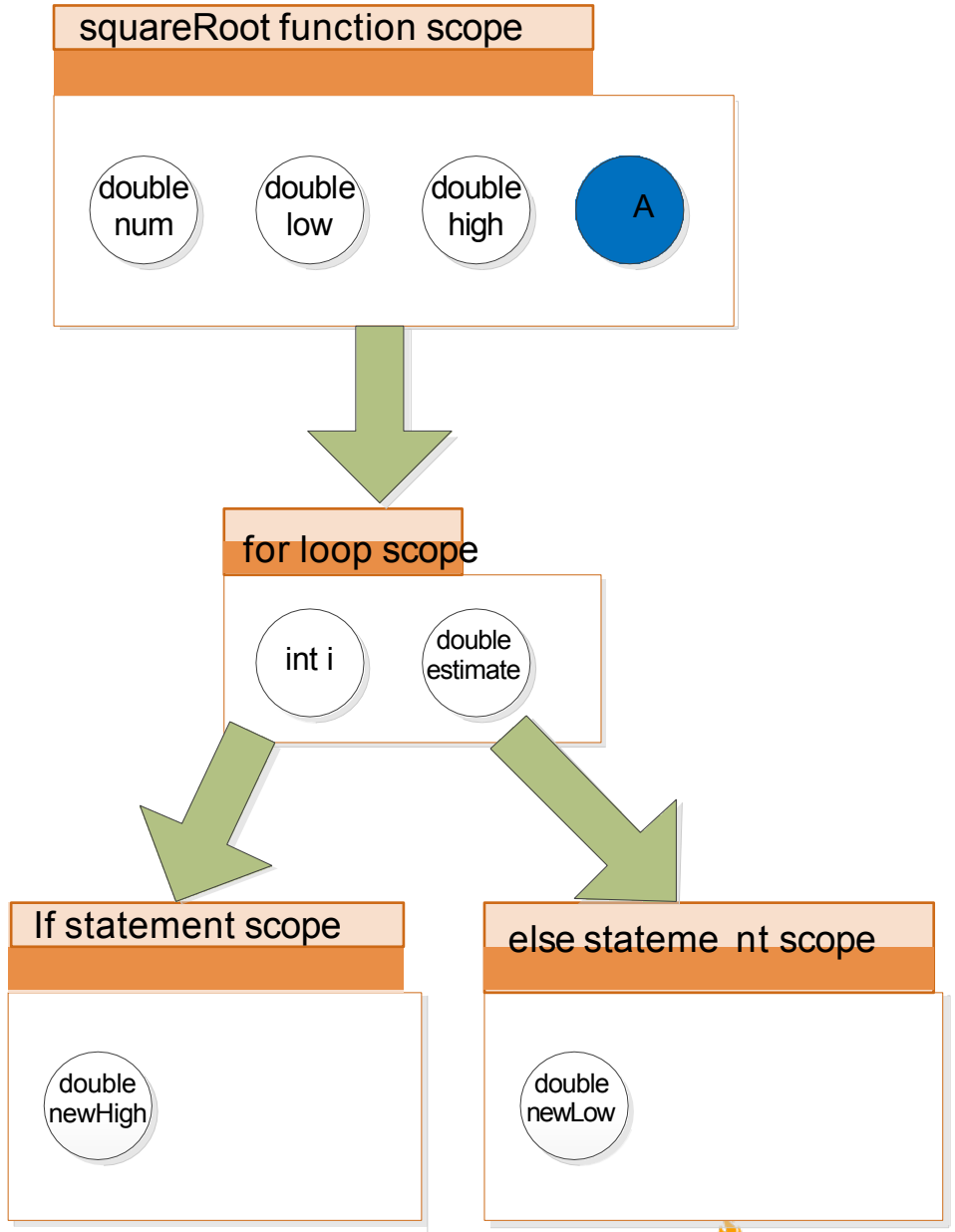
- Loops and if/else statements also have their own scopes



– Loop counters are in the same scope as the body of the for loop

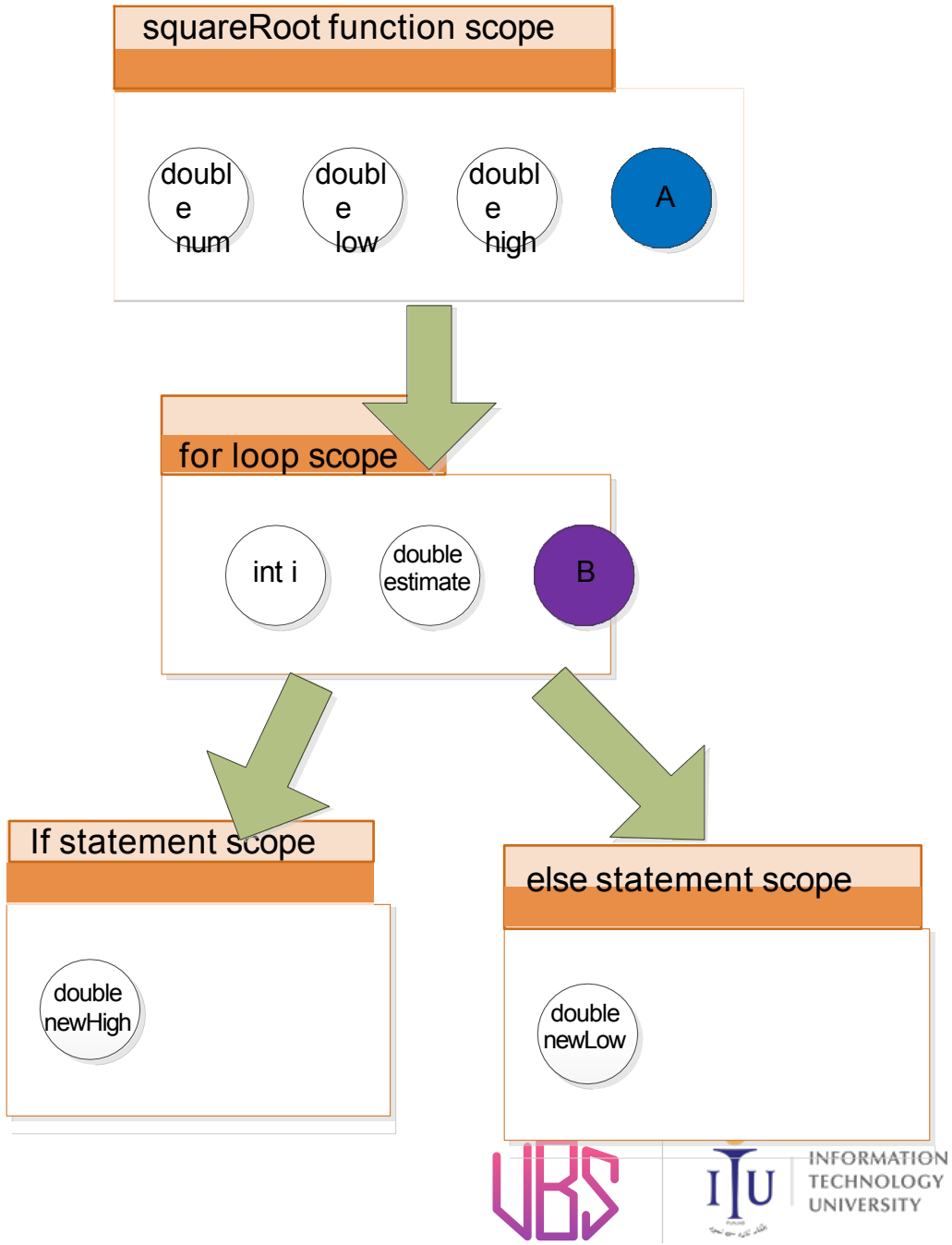
```
double squareRoot(double num) {
  double low = 1.0;
  double high = num;
  for (int i = 0; i < 30; i = i + 1) {
    double estimate = (high + low) / 2;
    if (estimate*estimate > num) {
      double newHigh = estimate;
      high = newHigh;
    } else {
      double newLow = estimate;
      low = newLow;
    }
  }
  // A
  return estimate; // ERROR
}
```

- Cannot access variables that are out of scope

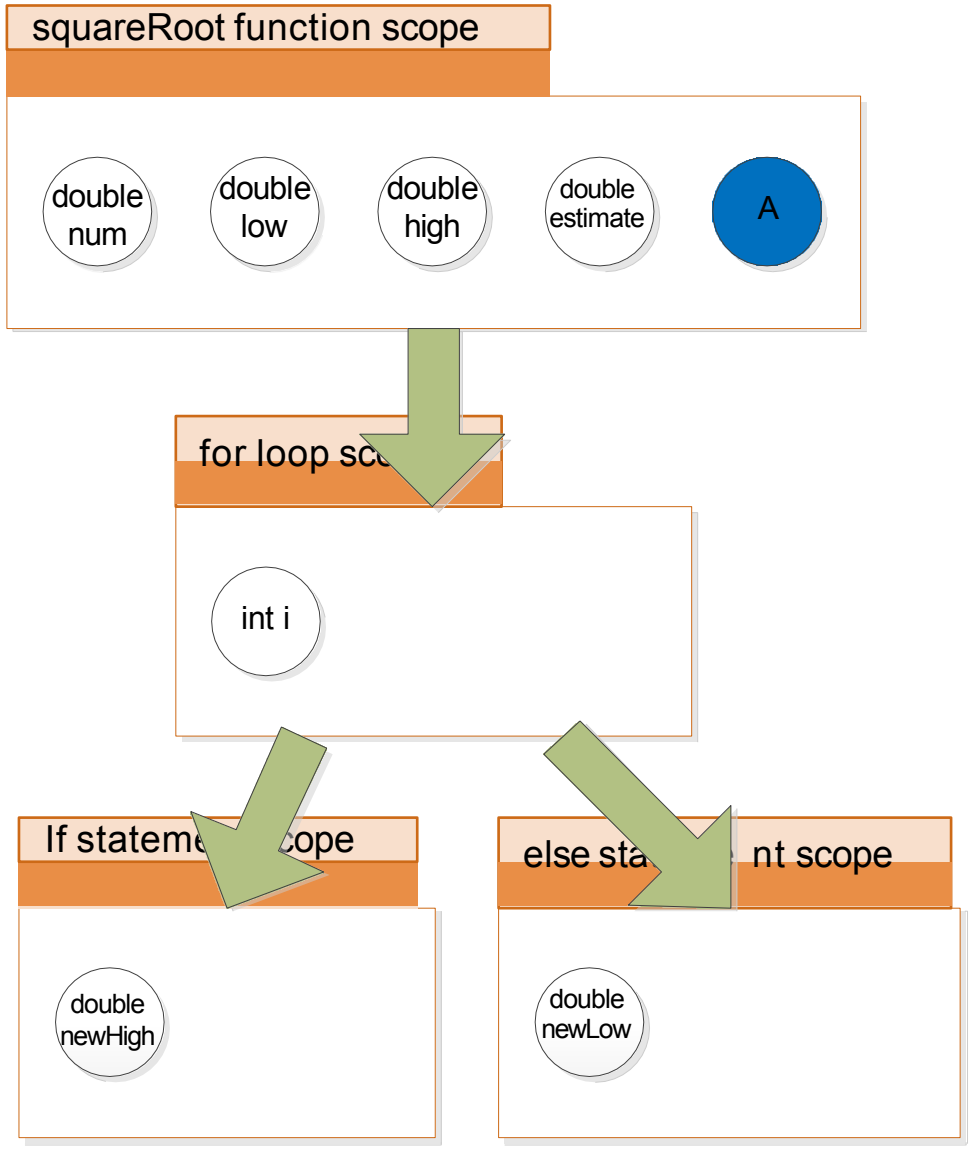


```
double squareRoot(double num) {
  double low = 1.0;
  double high = num;
  for (int i = 0; i < 30; i = i + 1) {
    double estimate = (high + low) / 2;
    if (estimate*estimate > num) {
      double newHigh = estimate;
      high = newHigh;
    } else {
      double newLow = estimate;
      low = newLow;
    }
    if (i == 29)
      return estimate; // B
  }
  return -1; // A
}
```

- Cannot access variables that are out of scope
- Solution 1: move the code



```
double squareRoot(double num) {
  double low = 1.0;
  double high = num;
  double estimate;
  for (int i = 0; i < 30; i = i + 1) {
    estimate = (high + low) / 2;
    if (estimate*estimate > num) {
      double newHigh = estimate;
      high = newHigh;
    } else {
      double newLow = estimate;
      low = newLow;
    }
  }
  return estimate; // A
}
```



- Cannot access variables that are out of scope
- Solution 2: declare the variable in a higher scope

```
void increment(int a) {
  a = a + 1;
  cout << "a in increment " << a << endl;
}

int main() {
  int q = 3;
  increment(q); //  does  nothing
  cout << "q in main " << q << endl;
}
```

```
void increment(int a) {
  a = a + 1;
  cout << "a in increment " << a << endl;
}

int main() {
  int q = 3;
  increment(q); //  does  nothing
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **3**

Pass by value vs by reference

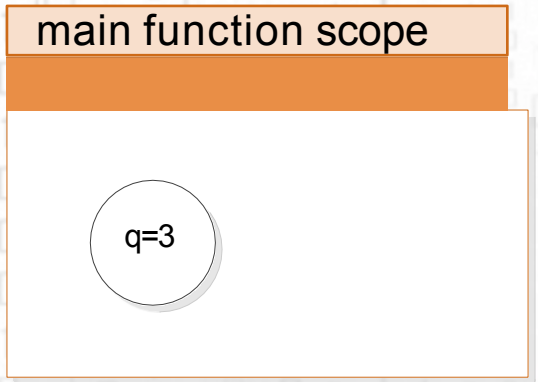
- So far we've been passing everything by value – makes a copy of the variable; changes to the variable within the function don't occur outside the function

```
// pass-by-value
void increment(int a) {
  a = a + 1;
  cout << "a in increment " << a << endl;
}

int main() {
  int q = 3;
  increment(q); //  does  nothing
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **3**

Pass by value vs by reference

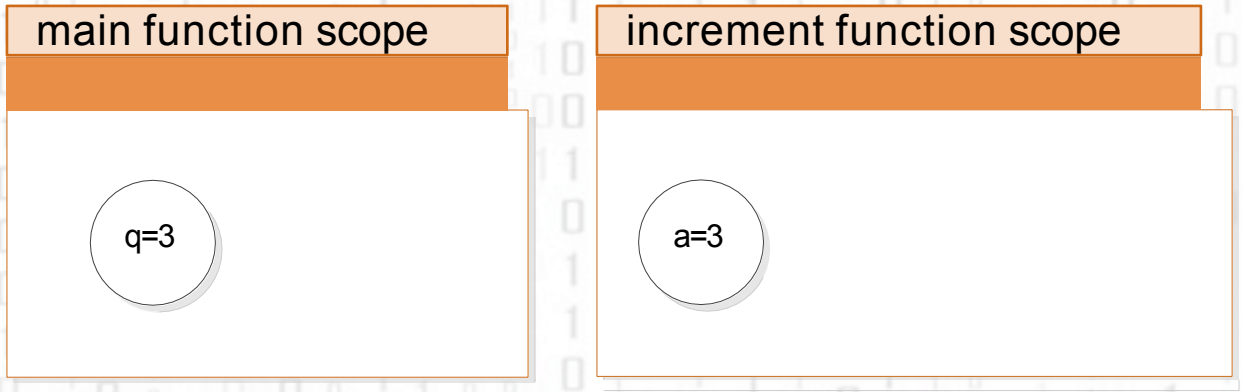



```
// pass-by-value
void increment(int a) {
  a = a + 1;
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3; // HERE
  increment(q); // does nothing
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **3**

Pass by value vs by reference

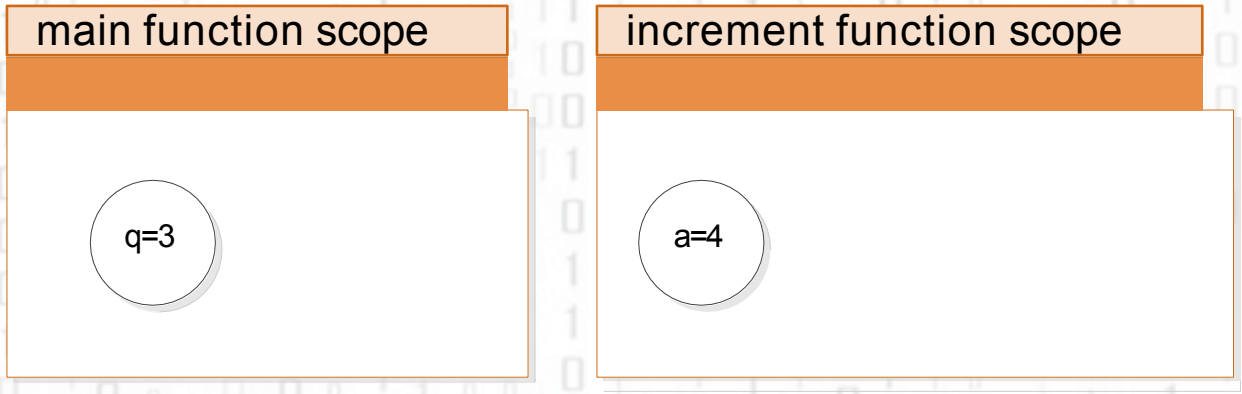



```
// pass-by-value
void increment(int a) { // HERE
  a = a + 1;
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3;
  increment(q); //  does  nothing
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **3**

Pass by value vs by reference




```
// pass-by-value
void increment(int a) {
  a = a + 1; // HERE
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3;
  increment(q); //  does  nothing
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **3**

Pass by value vs by reference

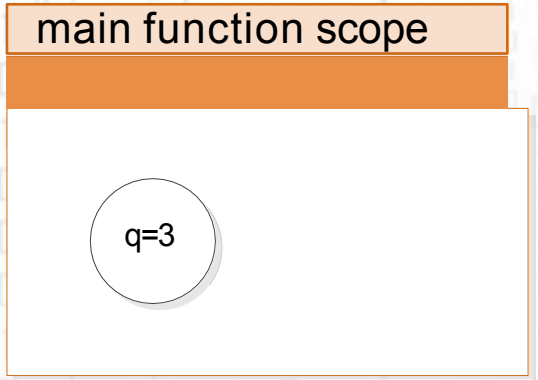
- If you want to modify the original variable as opposed to making a copy, pass the variable by reference (**int &a** instead of **int a**)

```
// pass-by-value
void increment(int &a) {
  a = a + 1;
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3;
  increment(q); // works
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **4**

Pass by value vs by reference

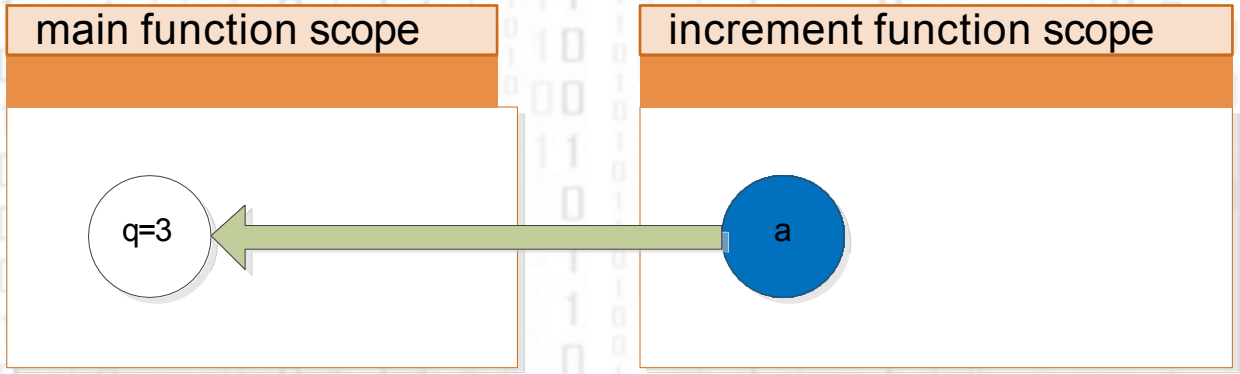



```
// pass-by-value
void increment(int &a) {
  a = a + 1;
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3; // HERE
  increment(q); // works
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **4**

Pass by value vs by reference

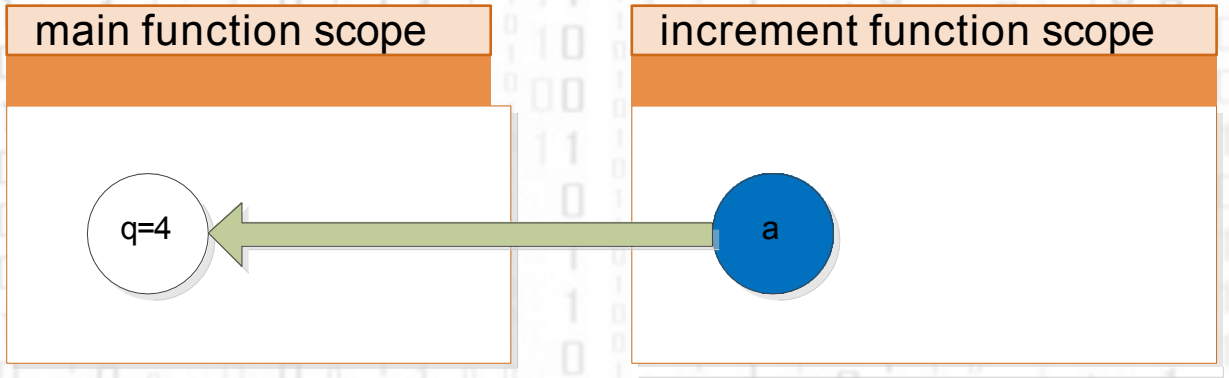



```
// pass-by-value
void increment(int &a) { // HERE
  a = a + 1;
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3;
  increment(q); // works
  cout << "q in main " << q << endl;
}
```

Output

a in increment 4
q in main **4**

Pass by value vs by reference




```
// pass-by-value
void increment(int &a) {
  a = a + 1; // HERE
  cout << "a in increment " << a << endl;
}
int main() {
  int q = 3;
  increment(q); // works
  cout << "q in main " << q << endl;
}
```

Output

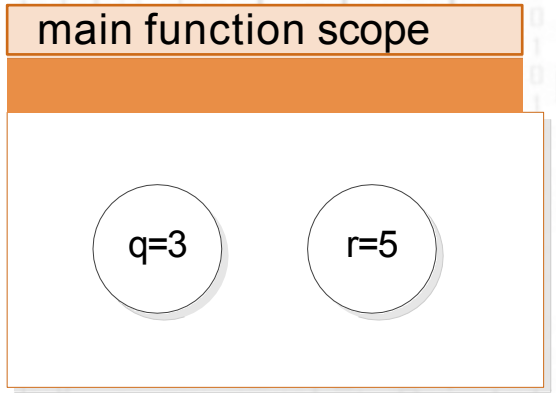
a in increment 4
q in main **4**

Implementing Swap

```
void swap(int &a, int &b) {
  int t = a;
  a = b;
  b = t;
}
int main() {
  int q = 3;
  int r = 5;
  swap(q, r);
  cout << "q " << q << endl; // q 5
  cout << "r " << r << endl; // r 3
}
```

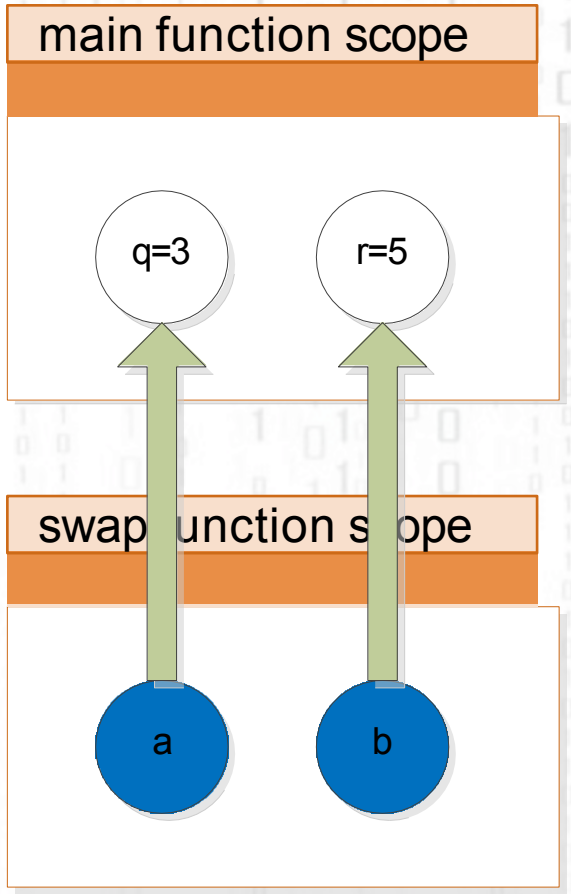
Implementing Swap

```
void swap(int &a, int &b) {
  int t = a;
  a = b;
  b = t;
}

int main() {
  int q = 3;
  int r = 5; // HERE
  swap(q, r);
  cout << "q " << q << endl; // q 5
  cout << "r " << r << endl; // r 3
}
```



Implementing Swap

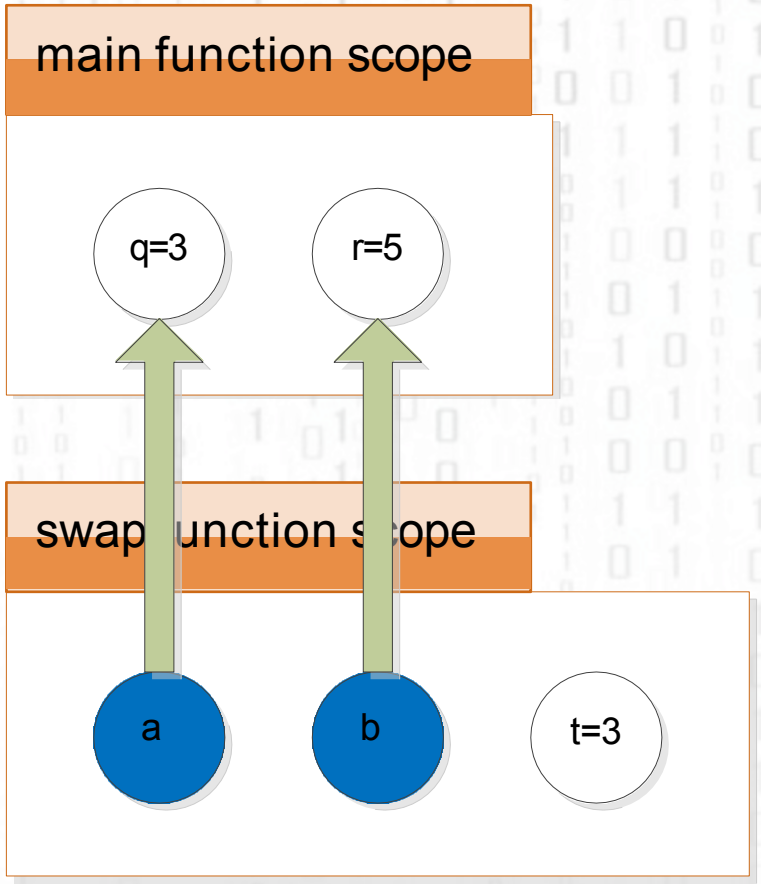
```
void swap(int &a, int &b) { // HERE
  int t = a;
  a = b;
  b = t;
}
int  main()
  {  int q =
  3;  int r =
  5;   swap(q,
  r);
  cout << "q " << q << endl; // q 5
  cout << "r " << r << endl; // r 3
}
```



INFORMATION TECHNOLOGY UNIVERSITY

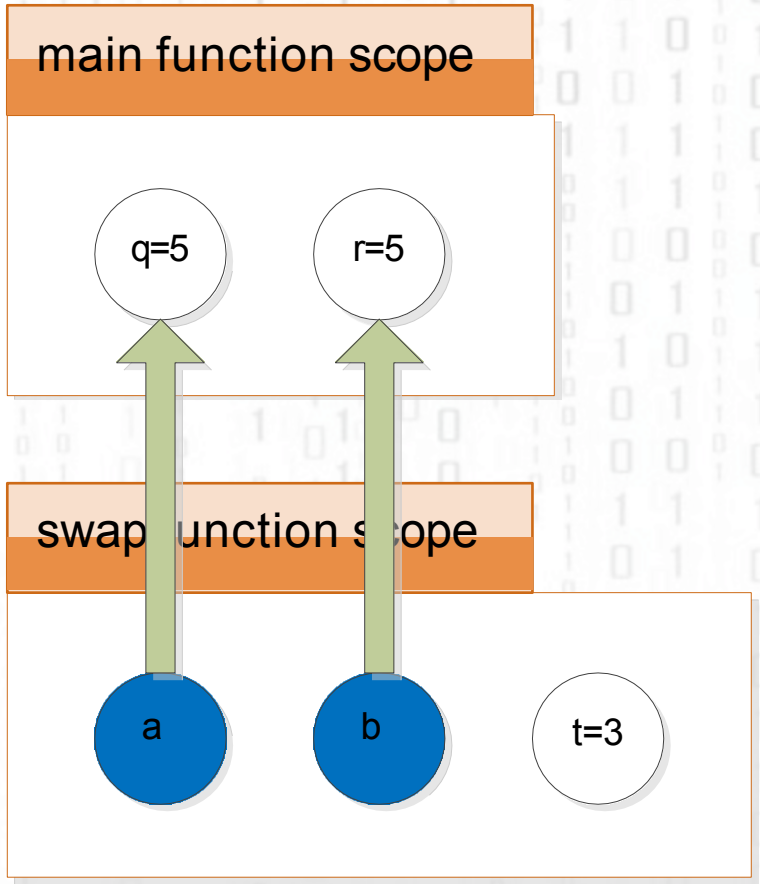
Implementing Swap

```
void swap(int &a, int &b) {
  int t = a; // HERE
  a = b;
  b = t;
}
int  main()
  {  int  q =
  3;  int r =
  5;  swap(q,
  r);
  cout << "q " << q << endl; // q 5
  cout << "r " << r << endl; // r 3
}
```



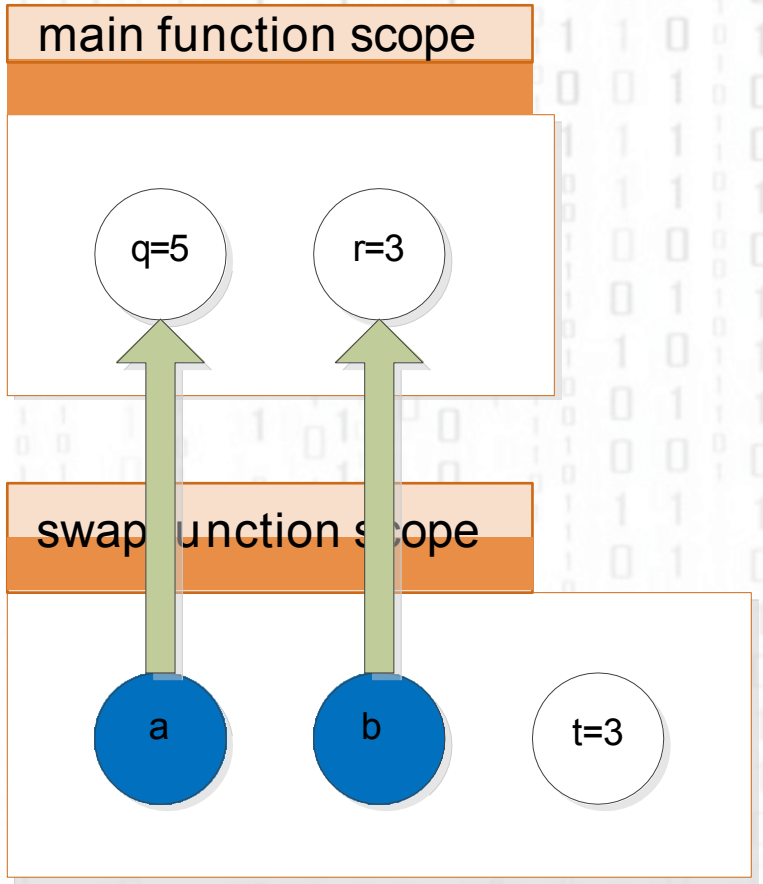
Implementing Swap

```
void swap(int &a, int &b) {
  int t = a;
  a = b; // HERE
  b = t;
}
int  main()
  {  int q =
  3;  int r =
  5;  swap(q,
  r);
  cout << "q " << q << endl; // q 5
  cout << "r " << r << endl; // r 3
}
```



Implementing Swap

```
void swap(int &a, int &b) {
  int t = a;
  a = b;
  b = t; // HERE
}
int  main()
  {  int q =
  3;  int r =
  5;  swap(q,
  r);
  cout << "q " << q << endl; // q 5
  cout << "r " << r << endl; // r 3
}
```



Returning multiple values

- The return statement only allows you to return 1 value. Passing output variables by reference overcomes this limitation.

```
int divide(int numerator, int denominator, int &remainder) {
  remainder = numerator % denominator;
  return numerator / denominator;
}

int main()
  {  int num =
  14; int den =
  4; int rem;
  int result = divide(num, den, rem);
  cout << result << "*" << den << "+" << rem << "=" << num << endl;
  // 3*4+2=12
}
```