CS1010E Topic 4: Pointers and Functions

Siau-Cheng KHOO
Block COM2, Room 04-11, +65 6516 6730
www.comp.nus.edu.sg/~khoosc
khoosc@nus.edu.sg

Semester II, 2017/2018

Revision: Repetition Statements

What are the values of variables n and i at line 7?

```
1 #include <stdio.h>
2 int f1(int i);

3 int main(void) {
4   int n = 1, i;
5   for (i = 4; i > 0; i--) {
6     n = f1(n);
    }
7   printf("%d\n", n);
8   return 0;
  }

9 int f1(int i) {
10  return 2*i;
  }
```

i =

n =

Revision: Repetition Statements

```
 1 #include <stdio.h>
 2 int f1(int i);
 3
 4 int main(void) {
 5     int n = 1, i = 1, j = 10;
 6     for (  ; i < j; i+=2, j-=2) {
 7       n = f1(n);
 8     }
 9     printf("%d %d %d\n", i, j, n);
10     return 0;
11 }
12
13 int f1(int i) {
14   return 2*i;
15 }
```

What are the values of variables n, i at j in line 9?

```
i =
j =
n =
```

Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

Question: How to reuse code?

```
 1 #include <stdio.h>
 2
 3 int main(void) {
 4     int x=1, y=2, z=3 ;
 5     int a=4, b=5, c=6, temp ;
 6
 7     temp = x ; x = a ; a = temp ;
 8
 9     temp = y ; y = b ; b = temp ;
10
11     temp = z ; z = c ; c = temp ;
12
13     printf("%d %d %d.\n", x, y, z) ;
14
15     return 0 ;
16 }
```

Question: How to reuse code?

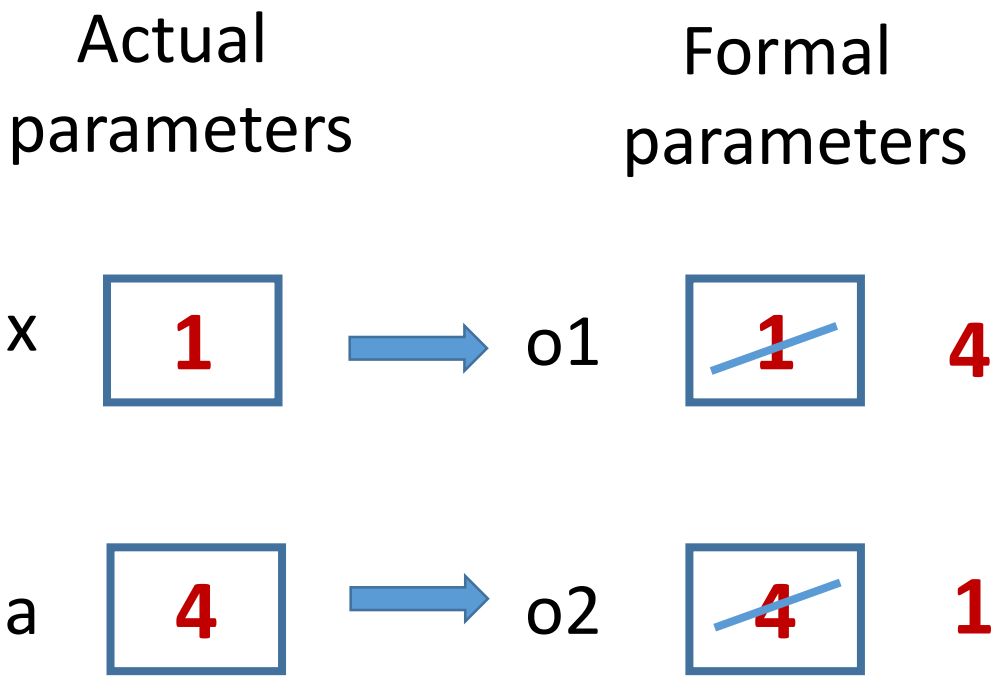
```
1 #include <stdio.h>
 2
 3 int main(void) {
 4     int x=1, y=2, z=3 ;
 5     int a=4, b=5, c=6, temp ;
 6
 7     temp = x ; x = a ; a = temp
 8
 9     temp = y ; y = b ; b = temp
10
11     temp = z ; z = c ; c = temp
12
13     printf("%d %d %d.\n", x, y,
14
15     return 0 ;
16 }
```

```
int main(void) {
    int x=1, y=2, z=3 ;
    int a=4, b=5, c=6 ;

    swap(x,a) ;  swap(y,b) ;  swap(z,c) ;
    printf("%d %d %d.\n", x, y, z) ;
    return 0 ;
}
void swap(int o1, int o2) {
    int temp ;
    temp = o1 ; o1 = o2 ; o2 = temp ;
}
```

Question: How to reuse code?

```
int main(void) {
    int x=1, y=2, z=3 ;
    int a=4, b=5, c=6 ;

    swap(x,a) ;  swap(y,b) ;  swap(z,c) ;
    printf("%d %d %d.\n", x, y, z) ;
    return 0 ;
}
void swap(int o1, int o2) {
    int temp ;
    temp = o1 ; o1 = o2 ; o2 = temp ;
}
```

Memory Snapshot



Call-By-Value

- The function call is typically **call-by-value** or **reference-by-value**.
- The **value** of the actual parameters during function call is passed to the function and is used as the **value** of the corresponding formal parameters.
- In general, a C function **cannot change** the value of an **actual** parameter
- **Question**: How would a function change the value of some variables **not declared** in its definition?

Call by Reference

- Exceptions occur when the actual parameters are arrays (discussed after the one-week break) or pointers (discussed later today)
- These exceptions generated a **call-by-reference** or a **reference-by-address**.

Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

Storage Class and Scope

- So far, we have always declared variables within a `main` function and within programmer/user-defined functions.
- We can also define a variable **before** the `main` function.
- It is important to be able to determine the **scope** of a function or a variable.
- Scope refers to the portion of the program in which it is valid to reference the function or variable.

Storage Class and Scope

- Scope is also sometimes defined in terms of the portion of the program in which the function or variable is **visible** or **accessible**.

- Because the scope of a variable is directly related to its **storage class**:
  - ~~automatic~~
  - external
  - static
  - ~~register~~

Local Variables

- **Defined within** a function, and thus include the formal parameters and any other variables declared in the function.

- Can be accessed only I the function that defines it.

- Usually has a value when its function is being executed, but its value is not retained when the function is completed.

Global Variables

- Defined **outside** the `main` function or other programmer-defined functions

- It is **defined outside of all** functions, and it can be **accessed by any function** within the program.

- To reference a global variable, some compilers require that the declaration within the function include the keyword **`extern`** before the type designation to tell the computer to look outside the function for the variable.

Storage Class and Scope

```
#include <stdio.h>
int count = 0 ;
. . .
int main(void) {

        int x, y, z ;
        . . .
```

```
int calc(int a, int b) {
    int x ;
    extern int count ;
    . . .
}
void check(int sum) {
    extern int count ;
    . . .
}
```

The variable **count** is a **global** variable that can be referenced by the functions `calc` and `check`.

Local variables: `x`, `y`, `z` in `main`; `a`, `b`, `x` in `calc`; `sum` in `check`; each referenced in their respective function.

Storage classes -- Static

- `static` is the default storage class for global variables.
- These 2 both have a static storage class

```
#include <stdio.h>
static int count = 0 ;
int road = 0 ;
. . .
int main(void) {
    printf("%d\n", count);
    printf("%d\n", road);
}
```

Storage classes – Static

- `static` can also be used to define a variable **within** a function. Such variable is then **initialised at compilation time** and **retains its value between calls**.
- Because it is initialised at compilation time, the initialisation value must be a constant.

```
int main(void) {
    func3(); func3(); func3();
    printf("\n") ;
    return 0;
}
void func3() {
    static int i = 0 ;
    printf("%d ", i++) ;
}
```

Style

- The memory assigned to an external variable is retained for the duration of the program.
- Although an external variable can be referenced from a function using the proper declaration, **using global variables is generally discouraged**.
- In general, **parameters are preferred** for transferring information to a function because the parameter is evident in the function prototype, whereas the external variable is not visible in the function prototype.
- The use of global variables should be avoided whenever possible; and **should not be used in your programs written for CS1010E**.

Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

Question 2: How to reuse code?

```
 3 int main(void) {
 4     int x, y, min, max ;
 5
 6     printf("Enter two integers: ") ;
 7     scanf("%d %d", &x, &y) ;
 8
 9     if (x < y) {
10         min = x ; max = y ;
11     } else {
12         min = y ; max = x ;
13     }
14
15     printf("minimum is %d, maximum is %d.\n", min, max) ;
16     return 0 ;
17 }
```

Question 2: How to reuse code?

```
 3 int main(void) {
 4     int x, y, min, max ;
 5
 6     printf("Enter two integers: ") ;
 7     scanf("%d %d", &x, &y) ;
 8
 9     if (x < y) {
10         min = x ; max = y ;
11     } else {
12         min = y ; max = x ;
13     }
14
15     printf("minimum is %d, maximum is %d.\n", min, max) ;
16     return 0 ;
17 }
```

```
int minimax(int a, int b) {
    int min, max ;
    if (a < b) {
        min = a ; max = b ;
    } else {
        min = b ; max = a ;
    }
    return min ;
    return max ;
}
```

Function Returns

- A function takes zero or more formal parameters as inputs
- It returns **zero or one** value.
- A **return** statement **terminates** the execution of a function **immediately** and returns flow control back to the caller.
- How would a function **return more than one value** to its callers?

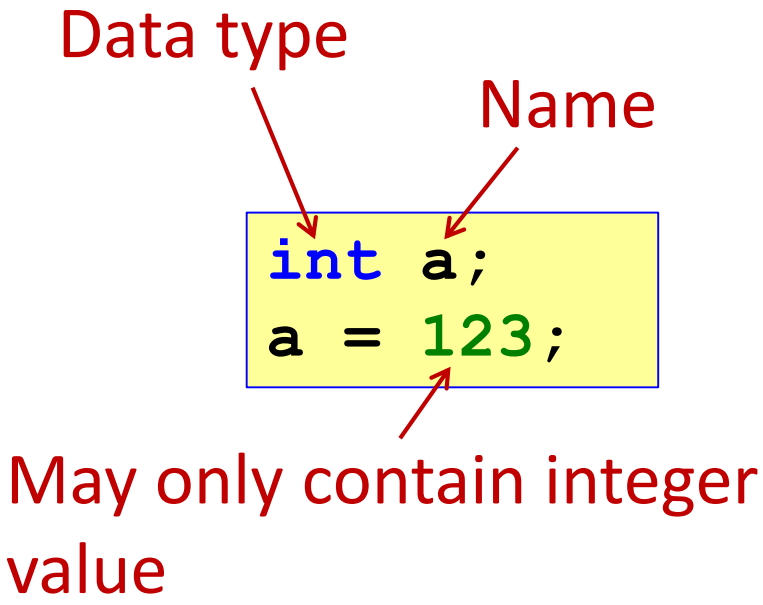
```
int minimax(int a, int b) {
    int min, max ;
    if (a < b) {
        min = a ; max = b ;
    } else {
        min = b ; max = a ;
    }
    return min ;
    return max ;
}
```

Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

Variable and Its Address

- (Recall) A **variable** has a unique **name** (identifier) in the function it is declared; it belongs to some **data type**, and it can contain a **value** of that type.
- A variable occupies some space in the memory, and so it has an **address**.

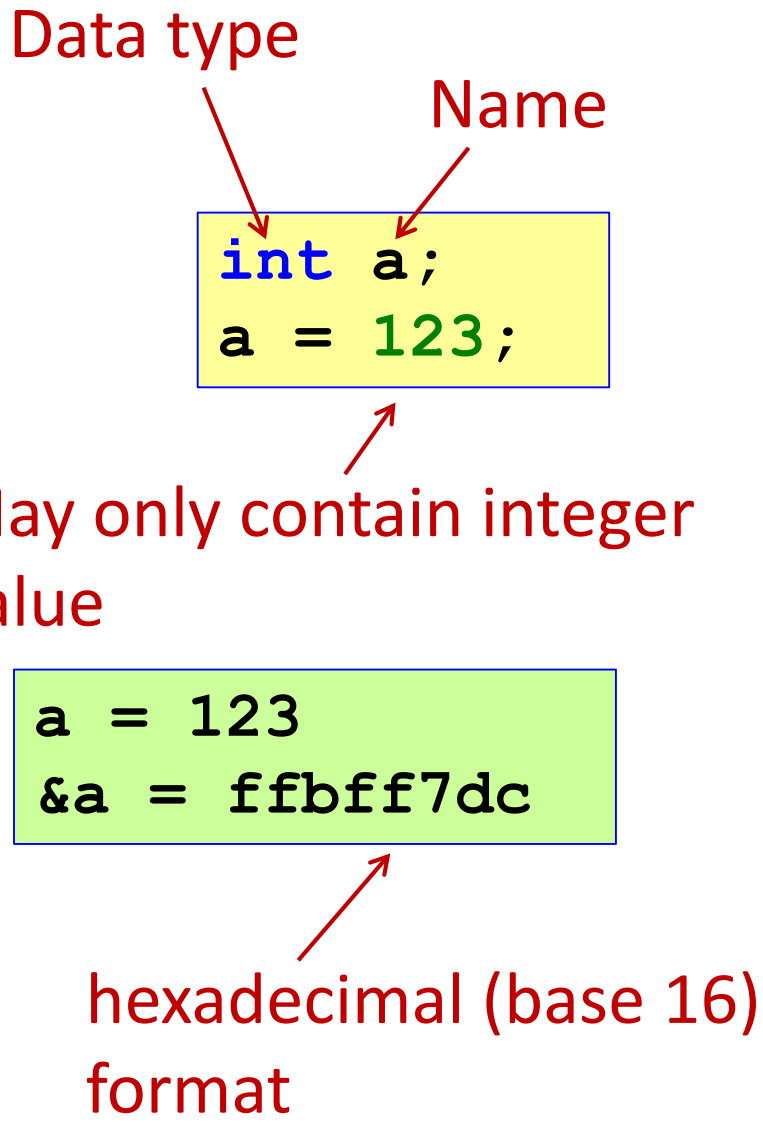


Variable and Its Address

- A variable occupies some space in the memory, and so it has an **address**.
- You may refer to the address of a variable by using the **address operator &**

```
int a = 123;
printf("a = %d\n", a);
printf("&a = %p\n", &a);
```

- %p is used as the conversion specifier for address



Variable and Its Address

- The address of a variable varies from run to run, as the system allocates any free memory to the variable

```
#include <stdio.h>

int main(void) {
        int a = 123;
        int *b = &a ;

        printf("a = %d\n", a);
        printf("&a = %p\n", &a);
        printf("*&a = %d\n", *&a) ;

        return 0;
}
```

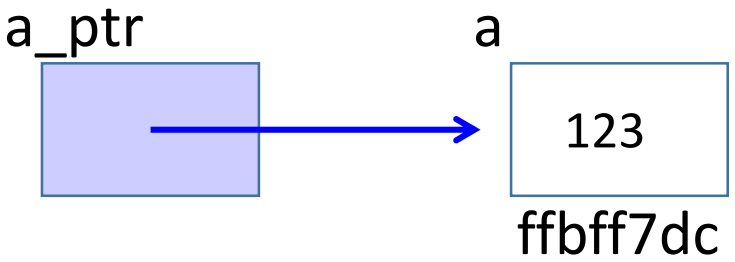
```
khoosc@suna0:~/beta/c/5[1011]$ a.out
a = 123
&a = ffbff208
*&a = 123
khoosc@suna0:~/beta/c/5[1012]$ a.out
a = 123
&a = ffbff238
*&a = 123
khoosc@suna0:~/beta/c/5[1013]$ a.out
a = 123
&a = ffbff318
*&a = 123
```

Pointer

- A variable that **contains the address** of another variable is called a pointer variable, or simply, a **pointer**.

Assuming that variable **a** *is located at address* ffbff7dc.

- Variable `a_ptr` is said to be **pointing** to variable a.
- If the address of a is immaterial, we simply draw an arrow from the blue box to the variable it points to.



Declaring a Pointer

Syntax:

```
type *pointer_name;
```

- pointer_name is the name (identifier) of the pointer
- type is the data type of the variable this pointer may point to

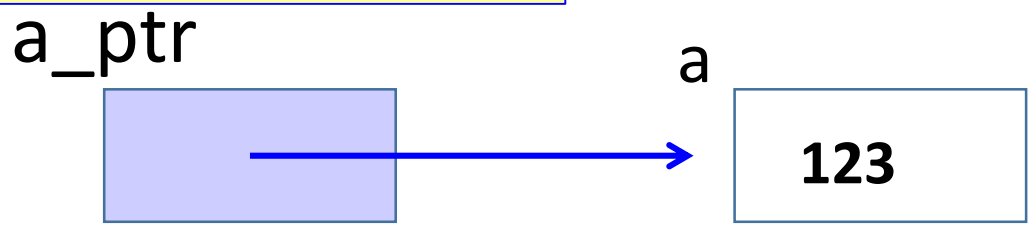
- Example: The following statement declares a pointer variable a_ptr which may point to any int variable
- Good practice to name a pointer with suffix _ptr or _p

```
int *a_ptr;
```

Assigning Value to a Pointer

- Since a pointer contains an address, only addresses may be assigned to a pointer
- Example: Assigning address of a to a_ptr

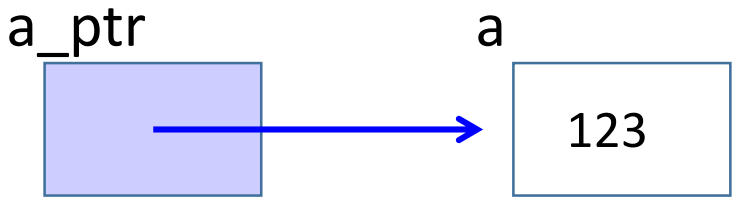
```
int a = 123;
int *a_ptr; // declaring an int pointer

a_ptr = &a;
```



- We may initialise a pointer during its declaration:

```
int a = 123;
int *a_ptr = &a; // initialising a_ptr
```

Accessing Variable Through Pointer



- Once we make a_ptr points to a (as shown above), we can now access a directly as usual, or indirectly through a_ptr by using the indirection operator (also called dereferencing operator): *

```
printf("a = %d\n", *a_ptr);
```

=

```
printf("a = %d\n", a);
```

```
*a_ptr = 456;
```

is equivalent to

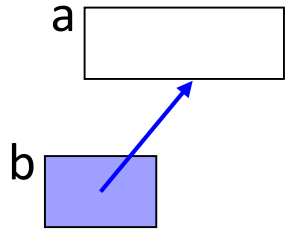
```
a = 456;
```

Hence, *a_ptr is synonymous with a

Example

```c
#include <stdio.h>

int main(void) {
  double a, *b;

  b = &a;
  *b = 12.34;
  printf("%f\n", a);

  return 0;
}
```

Can you draw the picture?
What is the output?




```
12.340000
```

What is the output if the **printf()** statement is changed to the following?

```c
printf("%f\n", *b);
```

```
12.340000
```

```c
printf("%f\n", b);
```

Compile with warning

```c
printf("%f\n", *a);
```

Error

What is the proper way to print a pointer? (Seldom need to do this.)

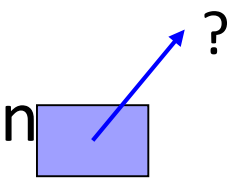
```c
printf("%p\n", b);
```

```
ffbff6a0
```

Common Mistake in Using Pointers

```c
#include <stdio.h>

int main(void) {
  int *n;

  *n = 123;
  printf("%d\n", *n);

  return 0;
}
```

What's wrong with this?
Can you draw the picture?



- Where is the pointer n pointing to?
- Where is the value 123 assigned to?
- Result: Segmentation Fault (core dumped)
  - Remove the file "core" from your directory. It takes up a lot of space!

Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

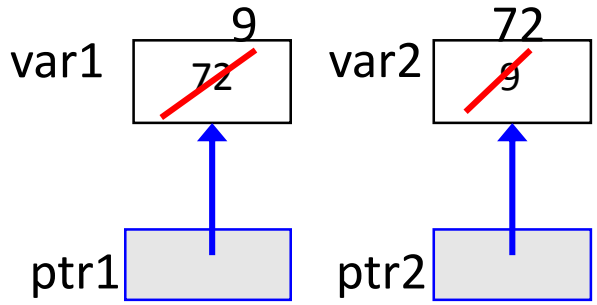
Function to Swap Two Variables

```c
#include <stdio.h>

void swap(int *, int *);

int main(void) {
   int var1, var2;

   printf("Enter two integers: ");
   scanf("%d %d", &var1, &var2);

   swap(&var1, &var2);

   printf("var1 = %d; var2 = %d\n", var1, var2);
   return 0;
}

void swap(int *ptr1, int *ptr2) {
   int temp;
   temp = *ptr1; *ptr1 = *ptr2; *ptr2 = t
}
```

Unit14_Swap_v3.c

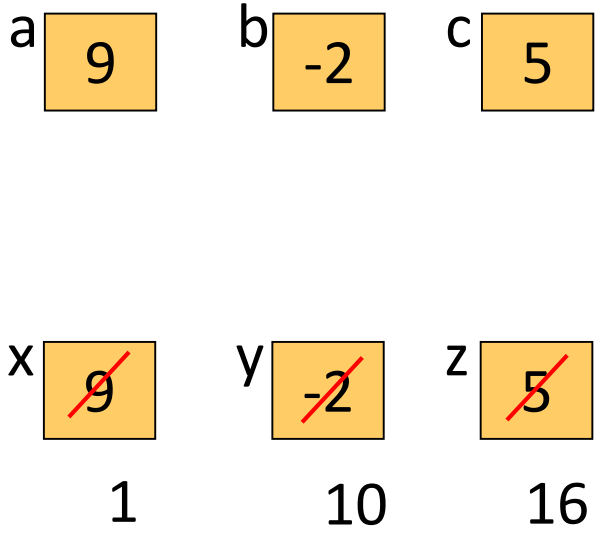
In main():

In swap():



Example (without Pointers)

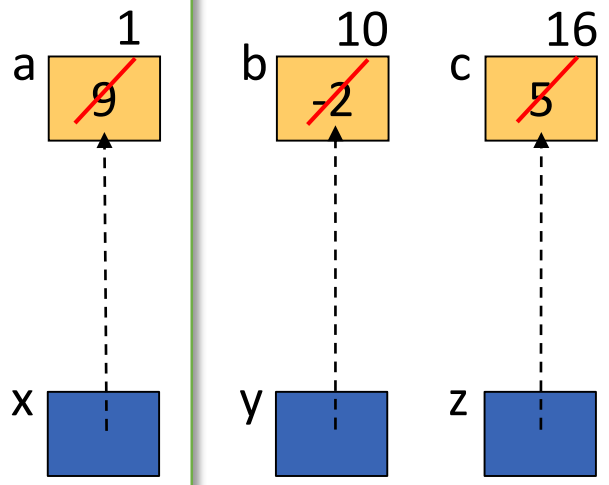
```
#include <stdio.h>
void f(int, int, int);

int main(void) {
    int a = 9, b = -2, c = 5;
    f(a, b, c);
    printf("a = %d, b = %d, c = %d\n", a, b, c);
    return 0;
}

void f(int x, int y, int z) {
    x = 3 + y;
    y = 10 * x;
    z = x + y + z;
    printf("x = %d, y = %d, z = %d\n", x, y, z);
}
```




```
x = 1, y = 10, z = 16
a = 9, b = -2, c = 5
```

Example (with Pointers)

```
#include <stdio.h>
void f(int *, int *, int *);

int main(void) {
    int a = 9, b = -2, c = 5;
    f(&a, &b, &c);
    printf("a = %d, b = %d, c = %d\n", a, b, c);
    return 0;
}
void f(int *x, int *y, int *z)
{
 ⟶ *x = 3 + *y;
 ⟶ *y = 10 * *x;
 ⟶ *z = *x + *y + *z;
 ⟶ printf("*x = %d, *y = %d, *z = %d\n", *x, *y, *z);
}
```

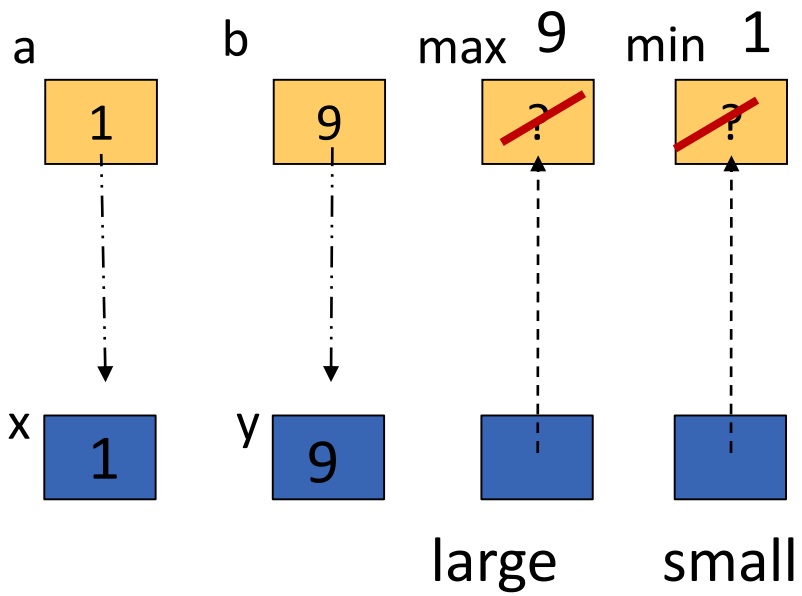


*x is a, *y is b, and *z is c!

```
*x = 1, *y = 10, *z = 16
a = 1, b = 10, c = 16
```

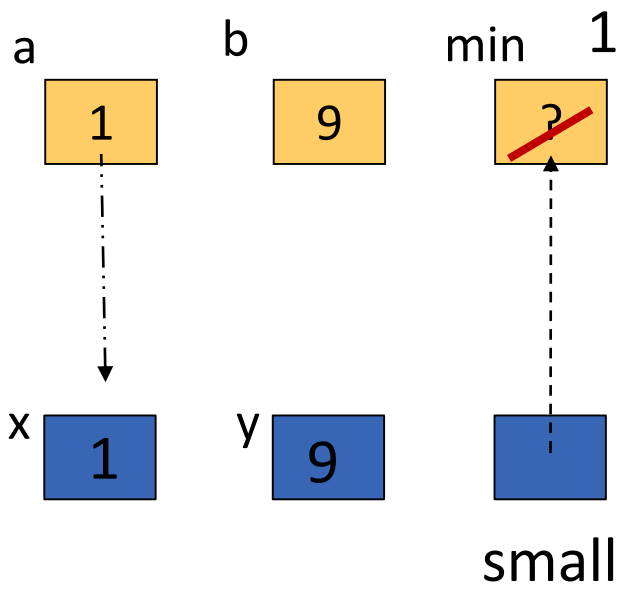
Function returning multiple values

```
void minimax(int, int, int *, int *) ;

int main(void) {
   int a = 1, b = 9, min, max ;

   minimax(a, b, &max, &min) ;

   printf("minimum is %d, maximum is %d.\n", min, max) ;
   return 0 ;
}

void minimax(int x, int y, int *large, int *small) {
   if (x < y) {
       *small = x ; *large = y ;
   } else {
       *small = y ; *large = x ;
   }
}
```



Function returning multiple values

```
int minimax(int, int, int *) ;

int main(void) {
   int a = 1, b = 9, min, max ;

   max = minimax(a, b, &min) ;

   printf("minimum is %d, maximum is %d.\n", min, max) ;
   return 0 ;
}

int minimax(int x, int y, int *small) {
   if (x < y) {
       *small = x ; return y ;
   } else {
       *small = y ; return x ;
   }
}
```



Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

Coercion of Function Arguments/Assignments

- (Recall) if we assign a value to a variable that has a different data type, then a conversion must occur during the execution of the statement.
- Sometimes the conversion can result in information being lost.

```
int a ;
. . .
a  = 12.8 ;
```

- Because a is defined as an integer, it cannot store a value with a nonzero decimal portion
- Therefore, a will contain the value 12 and **not** 12.8.
- Note: the value is truncated and not round up

Coercion via Numeric Conversion

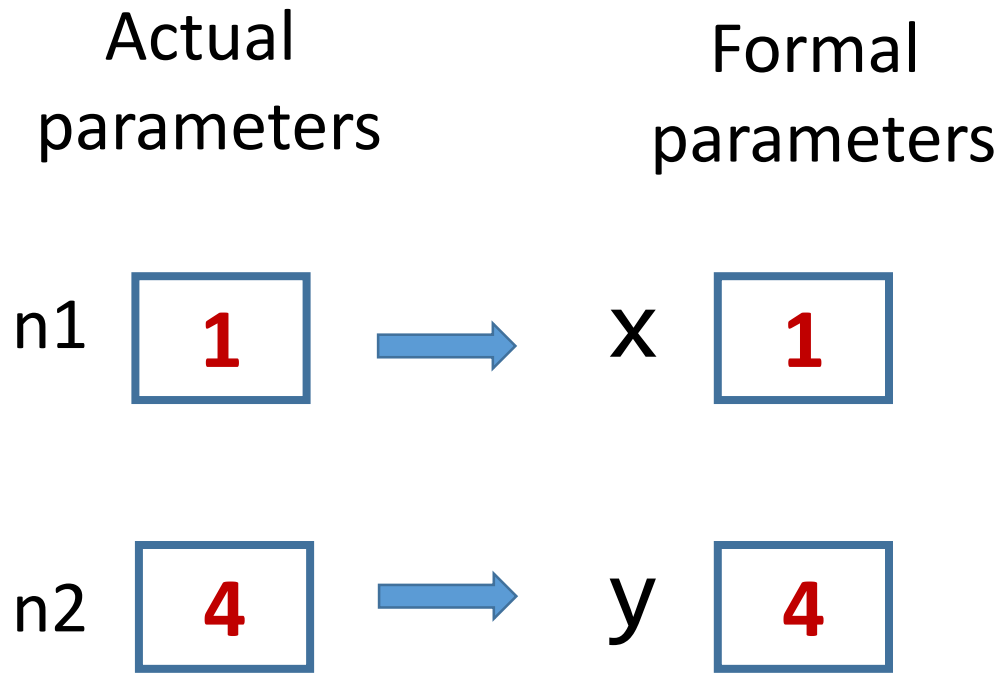
- To determine whether information will be lost from coercion, we use the following order (from high to low):

```
High: long double
      double
      float
      long integer
      integer
Low:  short integer
```

Coercion of Arguments

```
int max(int x, int y) {
    if (x < y) {
        return y ;
    } else {
        return x ;
    }
}
```

Memory Snapshot

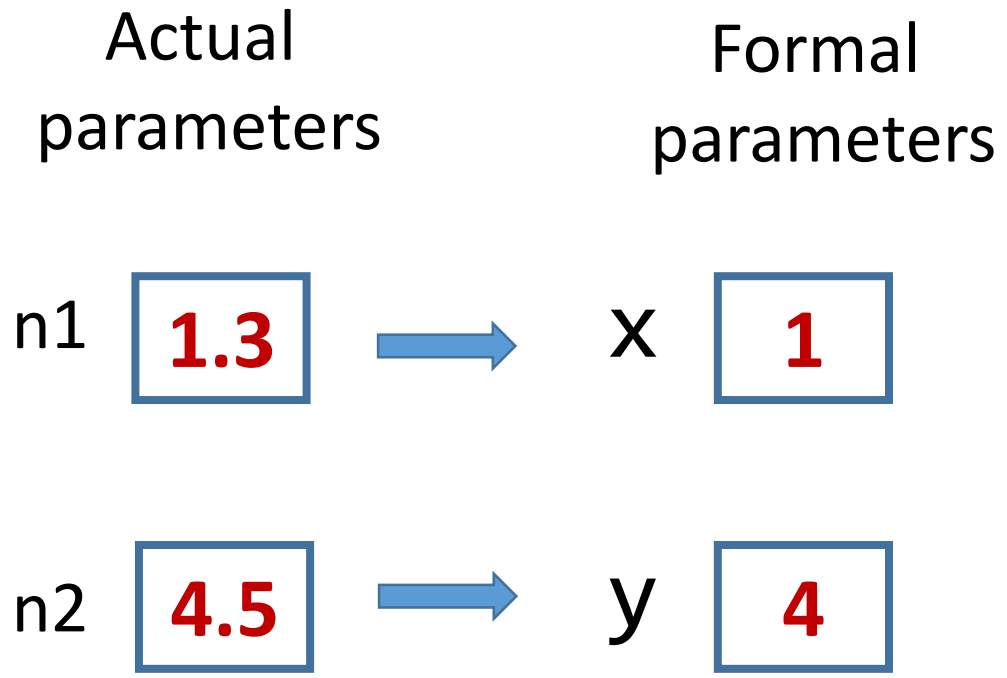



```
int n1 = 1; n2 = 4 ;
int larger ;

larger = max(n1,n2) ;
```

Coercion of Arguments

```
int max(int x, int y) {
    if (x < y) {
        return y ;
    } else {
        return x ;
    }
}
```

Memory Snapshot




```
double t1 = 1.3; n2 = 4.5 ;
double larger ;

larger = max(t1,t2) ;
```

Lecture Outline

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation

Random Numbers

- A sequence of random numbers is not defined by an equation; instead, it has certain characteristics that define it.

- These characteristics include the minimum and maximum values and the average.

- They are indicate whether the possible values are equally likely to occur or whether some values are more likely to occur than others.

Random Numbers

- Sequences of random numbers can be generated from experiments, such as tossing a coin, rolling a die, or selecting numbered balls.
- Sequences of random numbers can also be generated using the computer.
- Many engineering problems require the use of random numbers in the development of a solution.
- Sometimes the random numbers are used to develop a simulation of a complicated problem.

Random Numbers

- The simulation can be run over and over to analyse the results; each repetition represents a repetition of the experiment.
- We also use random numbers to approximate noise sequences.
- Eg: The static that we hear on radio is a noise sequence.
- If our test program uses an input data file that represents a radio signal, we may want to generate noise and add it to a speech signal or a music signal to provide a more realistic signal.

Random Numbers

- Engineering applications often require random numbers distributed between specified values.
- Eg: We may want to generate random integers between 1 and 500, or we may want to generate random floating-point values between 5 and -5.
- The random numbers generated are equally likely to occur.
- Random numbers that are equally likely to be any value in a specified set are also called **uniform random numbers,** or uniformly distributed.

Integer Sequences

- The Standard C library contains a function rand that generates a random integer between 0 and RAND_MAX, where RAND_MAX is a system-dependent integer defined in `stdlib.h`
- Thus, to generate and print a sequence of two random numbers, we could use this statement:
  - `printf("random numbers: %d %d \n", rand(), rand()) ;`
- Each time that a program containing this statement is executed, the same two values are printed, because the `rand` function generates integers in a specified sequence.
- Because this sequence eventually begins to repeat, it is sometimes called a **pseudo-random** sequence instead of a random sequence.

Integer Sequences

```
#include <stdio.h>
#include <stdlib.h>

int main (void) {
    int i ;
    printf(“5 random numbers generated:\n") ;

    for (i = 0; i < 4 ; i++) {
        printf("%d, ", rand()) ;
    }
    printf("and %d.\n", rand()) ;
    return 0;
}
```

```
khoosc@suna0:~/beta/e[1005]$ a.out
5 random numbers generated:
16838, 5758, 10113, 17515, and 31051.
khoosc@suna0:~/beta/e[1006]$ a.out
5 random numbers generated:
16838, 5758, 10113, 17515, and 31051.
```

Random Number Seed

- To cause a program to generate a new sequence of random values each time it is executed, we need to give a new **random-number seed** to the random-number generator.
- The function `srand` (from `stdlib.h`) specifies the seed for the random-number generator (default is 1).
- For each seed value, a new sequence of random numbers is generated by `rand`.
- The argument of the `srand` function is an unsigned integer that is used in computations that initializes the sequence; the seed value is not the first value in the sequence.

Random Number Seed

```
#include <stdio.h>
#include <stdlib.h>

int main (void) {
    int i ;
    unsigned int seed;

    printf("Enter a positive integer seed value: \n") ;
    scanf("%d", &seed) ;
    srand(seed) ;
    for (i = 0; i < 4 ; i++) {
        printf("%d, ", rand()) ;
    }
    printf("and %d.\n", rand()) ;
    return 0;
}
```

```
khoosc@suna0:~/beta/e[1009]$ a.out
Enter a positive integer seed value:
485
7346, 4586, 5434, 18611, and 30264.
khoosc@suna0:~/beta/e[1010]$ a.out
Enter a positive integer seed value:
3827
18310, 30886, 32432, 8578, and 2485.
```

Understanding Random Number Generator Prototype

- The rand function returns an integer and has no input:
  - `int rand(void) ;`

- The srand function returns no value and has an unsigned integer as an argument:
  - void srand(unsigned int) ;

Random Numbers within Specified Range

- Generate random integers between 0 and 7
  - `x = rand() % 8 ;`
- Generate random integers between -25 and 25
  - `x = rand() % 51 - 25;`
- Generate random integers between integers a and b
  - `x = rand() % . . . ;`

Random Numbers within Specified Range

- Generate random integers between 0 and 7
  - `x = rand() % 8 ;`
- Generate random integers between -25 and 25
  - `x = rand() % 51 - 25;`
- Generate random integers between integers a and b
  - `x = rand() % (b-a+1) + a ;`

Floating-point Sequences

- In many engineering problems, we need to generate random floating-point values in a specified interval [*a*,*b*].
- The computation to convert an integer between 0 and RAND_MAX to a floating-point value between *a* and *b* has three steps.
- **Step 1**: The value from the `rand` function is first divided by RAND_MAX to generate a floating-point value between 0 and 1.
- **Step 2**: The value obtained is then multiplied by (*b*-*a*) (from [*a*,*b*]) to give a value between 0 and (*b*-*a*).
- **Step 3**: The value obtained is then added to *a* to adjust it so that it will be between *a* and *b*.

Floating-Point Sequences

```
. . .
int main (void) {
    int i;
    double a = 3.4, b = 17.25 ;
    unsigned int seed;

    printf("Enter a positive integer seed value: \n") ;
    scanf("%d", &seed) ;
    srand(seed) ;
    for (i = 0; i < 4 ; i++) {
        printf("%.2f, ", rand_float(a,b)) ;
    }
    printf("and %.2f\n", rand  float(a,b)) ;
```

Floating-Point Sequences

```
int main (void) {
  . . .
 for (i = 0; i < 4 ; i++) {
        printf("%.2f, ", rand_float(a,b)) ;
    }
    printf("and %.2f\n", rand_float(a,b)) ;

    return 0;
}
double rand_float(double a, double b) {
    return ((double) rand() / RAND_MAX) * (b-a) + a;
}
```

Summary

- Functions for Code Reuse: Take One
- Storage Class and Scope
- Functions for Code Reuse: Take Two
- Pointers: Its Declaration and Its Use
- Using Pointers in Functions
- Function Arguments: Coercion Issues
- Special Function: Random Number Generation