References

CMPT 115/117 lecture slides

Notes written by Mark Eramian, Ian McQuillan, Michael Horsch, Lingling Jin, and Dmytro Dyachuk

Objectives

By the end of this lecture topic, you are expected to be able to

1. compare and distinguish the operators * and &
2. write pseudocode and C++ code to
    - declare a pointer
    - reference to the address of a variable
    - dereference a pointer
3. differentiate the use of * in function parameter (function header) and function argument (function call)
4. implement functions in C++ that use references as parameters and return type, as well as calling such functions

Variables

In C++, all variables have:

- A type: decided by the programmer.
- A name: decided by the programmer.
- A value: determined by the program.
- An address: determined at runtime by the OS.

Normal variable declaration

```
myNum ←7;
```

memory looks like

address	contents	
⋮	⋮	
4683952	…	
4683953	7	myNum
4683954	🟩	myNum
4683955	🟩	myNum
4683956	🟩	myNum
4683957	…	
⋮	⋮	

Uses for addresses

Advanced programming techniques that make use of addresses:

- Work with compound data without copying the data (e.g., arrays, records)
- Organize large collections
- Make some calculations more efficient

We'll see lots of examples of these! First we master the basics.

Three new concepts

1. A new kind of type: reference (also called “address”).
2. An operator (&) to acquire the address of a variable
3. An operator (*) to use the address of a variable

Warning

These seem easy at first, but these are the main source of program failure.

Finding the address of a variable

Address of...

In pseudocode and C++, if `var` is a variable name, then the expression `&var` is the address of that variable.

```
Integer myNum ← 7
print "the value stored in myNum is ", myNum
print "the address of myNum is ", &myNum
```

The value (7) is predictable from the program; the address is not!

Be careful!

In C++, the symbol & has a number of different meanings in other contexts!

Declaring pointers in pseudocode

A pointer is a variable that stores a reference.

```
Integer myNum             // This declares an integer
refToInteger myNumPtr     // This declares a pointer to an integer
```

- `myNumPtr` has a new kind of type: reference to an integer.
- The *only* kind of value we can put in `myNumPtr` is an address of an integer variable.
- We will use the same syntax for all our types in pseudocode: `refToX` stores a reference to an `X`.
- For example, `refToInteger`, `refToChar`, etc.. If we've created a record type called Date, it would be `refToDate`.

Assigning values to pointers in pseudocode

One way to initialize a pointer is to use & as follows:

```
Integer myNum
refToInteger myNumPtr

myNum ← 7
myNumPtr ← &myNum // This puts the address of myNum
                  // into the variable myNumPtr
```

- The variable `myNum` will contain 7, while the the variable `myNumPtr` will contain 4683953
  (or whatever address is given to the variable)
- Before initialization, a pointer contains garbage.

Using references

- A valid pointer contains the address of some data.
- The pointer "points to" the data.
- Following the pointer is called "de-referencing" the pointer.
- Problem: dereferencing has 2 related but distinct meanings.
    1. To refer to the value stored at the address (for use in normal calculations)
    2. To allow storage of data at the address (for use in assignment statements)

Warning

The meaning of a dereference depends on how it is used.

Using the address of a variable: getting a value

Using the value stored at an address

In pseudocode and C++, if `varPtr` contains a valid reference to a location, you can obtain the value stored in that location using the expression `*varPtr`.

```
Integer myNum ← 7
refToInteger myNumPtr ← &myNum

print "the value stored in myNum is", myNum
print "the value, again, is", *myNumPtr

myNum ← *myNumPtr + 1    // Pay attention to this line
```

Using the address of a variable: storing a value

Using the location referred to by a pointer

In pseudocode and C++, if `varPtr` contains a valid reference to a location, you can use the expression `*varPtr` on the Left Hand Side (LHS) of an assignment to refer to the location pointed to by the pointer.

```
Integer myNum ← 7
refToInteger myNumPtr ← &myNum

*myNumPtr ← *myNumPtr + 1  // Pay attention to this line
```

Tricky bits

```
Integer myNum ← 7
refToInteger myNumPtr ← &myNum
myNum      ← myNum      + 1
*myNumPtr ← *myNumPtr + 1
```

- Here, myNum and *myNumPtr have exactly the same uses.

  LHS: they both refer to the same location.

  RHS: they both refer to the same value (not a copy!)

- The real trick is to keep clear the difference between myNumPtr and *myNumPtr

Asymmetry in assignment statements

The left side of ← has different rules from the right side.

```
Integer myNum

myNum ← 7
myNum ← myNum + 1     // Pay attention to this line
```

- On the right side of ← `myNum` refers to a value.
- On the left side of ← `myNum` refers to a location.

Asymmetry in assignment statements

The left side of ← has different rules from the right side.

```
Integer myArray[10];

myArray[0] ← 7
myArray[1] ← myArray[0] + 1    // Pay attention to this line
```

- On the right side of ←: an expression is evaluated for its value.
- On the left side of ←: an expression is evaluated for its location, if the expression makes sense as a location.

Exercises 1

For each of the following types

- `Float`
- `Char`

Do the following:

1. Declare a pointer to a variable of the type
2. Initialize the pointer
3. Set a new value using `*`
4. Display the new value on the console using `print`

Pointers as parameters

Here’s a function:

```
Algorithm swap(a, b)
Pre: a :: refToInteger
     b :: refToInteger
     a, b contain valid references
Post: the contents of *a and *b
      are exchanged

  Integer temp ← *a
  *a ← *b
  *b ← temp
```

Two ways to use it

```
Integer x ← 5
Integer y ← -2

swap(&x, &y)

refToInteger ptr1 ← &y
refToInteger ptr2 ← &x

swap(ptr1, ptr2)
```

Swap example

```
Algorithm swap(a, b)
Pre: a :: refToInteger
     b :: refToInteger
     a, b contain valid references
Post: the contents of *a and *b
        are exchanged

  Integer temp ← *a
  *a ← *b
  *b ← temp
```

```
Integer x ← 3
Integer y ← 2
swap(&x,&y)
```

main memory

address	contents	
⋮	⋮	
4683953	3	x
4683954	🟩	x
4683955	🟩	x
4683956	🟩	x
4683957	2	y
4683958	🟨	y
4683959	🟨	y
4683960	🟨	y
⋮	⋮	

- What is stored in a and b?
- What is stored in x and y after `swap()` is called?

Pointers and functions

The reason for using references

Passing an address to a function makes a copy of the address, not a copy of the data being “pointed at.”

By passing in references to functions, functions have access to data outside the function scope.

This technique makes advanced uses of references very valuable!
We will see more of this!

Returning references in pseudocode

Returning references

A function may return a value of `refToX` where `X` is any type.

```
Algorithm findBigger(x, y)

Pre: x, y :: refToInteger are valid references
Post: no change to data
Return: the reference to the larger of *x, *y

    refToInteger temp
    if ($*x \geq *y$)
        temp $\leftarrow$ x
    else
        temp $\leftarrow$ y
    end if
    return temp
```

Returning References to Local Variables

Danger!

Never have a function return a reference which points to a local variable!

- Local variables are created when a function is called.
- Local variables are destroyed when a function returns.
- Returning an address of a local variable points to something just destroyed!

Exercises 2

Define a function that accepts 3 pointers, one each of

- `Float`
- `Char`
- `Integer`

Your function should display the values pointed to.

Exercises 3

Suppose you had a function with the following header:

```
Algorithm findBigger2(x, y, z)

Pre: x, y :: Integer
     z :: refToInteger, is a valid reference
Post: *z points to the larger value of x, y
Return: nothing
```

1. Complete the function by writing pseudocode for the body.
2. Give an example of calling this function; declare all variables you need!

Exercises 4

Find all the errors in the function:

```
Algorithm findBigger3(x, y)

    Integer temp $\leftarrow$ &x
    refToInt z $\leftarrow$ &temp
    if (*x $\geq$ *y)
        &temp $\leftarrow$ x
    else
        temp $\leftarrow$ *y
    end if
    return z
```

Pointers – in C++

- What we have seen about * and & is the same in pseudocode and C++.
- In pseudocode, we declare pointers using the prefix `refTo`.
- There is a good reason:
  - C++ reuses the symbol * for a third meaning!

Declaring Pointers in C++ is different

Pseudocode:

```
Integer myNum
refToInteger myNumPtr
```

C/C++:

```
int myNum;
int *myNumPtr;
```

- Suppose X is any type.
- Pseudocode: declare a pointer named ptr using refToX ptr
- C++: declare a pointer named ptr using X *ptr

Warning!!!

This is the third meaning of * in the context of references.

Pointers in C++

Here, the ∗ is being used to declare a pointer.

```
int myNum;                  This declares an integer
int *myNumPtr;              This declares a pointer to an integer

myNum = 7;
myNumPtr = &myNum;          This puts the address of myNum
                             into the variable myNumPtr
```

What does this look like in memory?

address	contents	
⋮	⋮	
4683953	7	myNum
4683954	🟩	myNum
4683955	🟩	myNum
4683956	🟩	myNum
4683957	4683953	myNumPtr
4683958	🟨	myNumPtr
4683959	🟨	myNumPtr
4683960	🟨	myNumPtr

Pointer dereferencing in C++

```
int myNum;
int *myNumPtr;

myNum = 7;
myNumPtr = &myNum;
*myNumPtr = *myNumPtr + 1;
cout << "The contents of myNum is" << myNum;
```

What does this look like in memory?

address	contents	
⋮	⋮	
4683953	7	myNum
4683954	🟩	myNum
4683955	🟩	myNum
4683956	🟩	myNum
4683957	4683953	myNumPtr
4683958	🟨	myNumPtr
4683959	🟨	myNumPtr
4683960	🟨	myNumPtr

Pointers and functions in C++

Pseudocode

```
Algorithm swap(a, b)
Pre: a :: refToInteger
     b :: refToInteger
     a, b contain valid references
Post: the contents of *a and *b
       are exchanged

   Integer temp ← *a
   *a ← *b
   *b ← temp
```

```
Integer x ← 3
Integer y ← 2
swap(&x,&y)
```

C++

```
void swap (int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

```
int x = 3;
int y = 2;
swap(&x,&y);
```

Returning pointers in C

```
Algorithm findBigger(x, y)

Pre: x, y :: refToInteger are valid references
Post: no change to data
Return: the reference to the larger of *x, *y

    refToInteger temp
    if (*x $\geq$ *y)
        temp $\leftarrow$ x
    else
        temp $\leftarrow$ y
    end if
    return temp
```

```c
int *findBigger(int *x, int *y){
    int *temp;
    if (*x >= *y)
        temp = x;
    else
        temp = y;
    return temp;
}
```

Arrays

- Remember "pass by reference" for arrays?
- Arrays in C++ are actually just pointers in disguise!
- When an array is passed to a function, C++ passes a pointer to the first element of the array.
- This way, the array does not have to be copied.
- Because the address is used, the function can change data outside the scope of the function.

Arrays as references

```
int findLargest (int someInts[], int size) {

    int largestSoFar = someInts[0];
    for (int i = 1 ; i < size; i++) {
        if ( someInts[i] > largestSoFar) {
            largestSoFar = someInts[i];
        }
    }
    return largestSoFar;
}
```

```
int ints[] = {2, 3, 5, 7};

large = findLargest(ints, 4);
cout << "Largest is " << large;
```

Exercises 5

Define a function in C++ that accepts 3 pointers, one each of

- `float`
- `char`
- `int`

Your function should display the values pointed to.

Exercises 6

Suppose you had a function with the following pseudocode header:

```
Algorithm findBigger2(x, y, z)

Pre: x, y :: Integer
     z :: refToInteger, is a valid reference
Post: *z points to the larger value of x, y
Return: nothing
```

1. Rewrite the header in C++.
2. Complete the function by writing C++ for the body.
3. Give an example of calling this function; declare all variables you need!

Exercises 7

Find and fix all the errors in the function:

```
void findBigger3(int x, int *y) {

    int temp = &x;
    int *z = &temp;
    if (*x >= *y) {
        &temp = x;
    }
    else {
        temp = *y;
    }
    return z;
}
```

Then demonstrate how to use your function; declare all variables you need.