The previous lecture explains that a function can see only its own frame. As a result, writing a swap function is difficult.

The swap function cannot see aee in line 1 nor bee in line 2.

The symbols aee in line 1 is different from the argument of the swap function because they occupy different memory space.

Fortunately, the designers of the C language has a solution for this: using pointers.

What is a pointer? A pointer is a variable whose value is an address.

Let’s consider an example. Suppose tee is an integer and its value is 5. It is stored in the stack memory. As explained earlier? Compilers and operating systems decide the addresses and the addresses are never zero.

For simplicity, I assign 100 as the address of tee.

The next line creates a pointer called pee. Pee is a pointer because there is an asterisk in front of pee.

By adding an asterisk, pee is a pointer. This means pee’s value is an address.

We also need to specify what type of information is stored at that address.

This example uses int as the type. That means the address stores an integer.

Adding & in front of tee means getting the address of tee. In this example, tee’s address is 100 and pee’s value is 100. For clarity, I add Aee in pee’s value to indicate that this is an address.

Again, please remember that as a programmer, you have no control of the addresses. If you run the program again, the address of tee may be different.

Please notice that we can initialize pee immediately after creating this pointer, as shown on the left side. We can also create the pointer first, and then assign tee’s address to pee, as shown on the right side.

The left side and the right side are exactly the same.

How do we use a pointer? There are two different ways, depending on whether asterisk is on the left hand side or the right hand side of the equal sign. The left hand side can also be written as L H S. .

The right hand side can also be written as R H S.

.

Either side has three steps and the first two steps are the same: The first step takes pee’s value as the address. The second step goes to that address.

The third step is different, depending on it is left hand side or right hand side. For left hand side, the third step modifies the value at that address. For right hand side, the third step reads the value stored at that address.

Let’s trace the code and understand how things work.

Suppose the program reaches line Cee. This line assigns tee’s address to pee’s value. As a result, pee’s value is 100.

Consider the program reaches line Dee. This line has three steps. The first step takes pee’s value and it is 100.

The second step goes to address 100.

The third step modifies the value at address 100 to -6. As a result, the value of tee becomes -6 now.

Next, let’s consider line ee. In this case, the asterisk is at the right hand side of the equal sign. The first two steps are the same.

The first step takes pee’s value as an address. The value is 100.

The second step goes to that address.

The third step reads the value from that address. The value of tee is -6.

Thus, line ee assigns -6 to the variable as.

Some people think pointers are confusing. This is understandable because the symbol asterisk has three different meanings:

The first way of using asterisk is to create a pointer. This must have a data type, such as int or double or char. This must have a data type in front of the asterisk.

The second way using asterisk is when it is at the left hand side of an assignment. When this occurs, the asterisk means the three steps explained earlier: 1. takes pee’s value and treat it as an address, 2. goes to that address, 3. modifies the value at that address.

The third way using asterisk is when it is at the right hand side of an assignment. When this occurs, the asterisk means the three steps explained earlier: 1. takes pee’s value and treat it as an address, 2. goes to that address, 3. reads the value at that address.

To make things even worse, there is the fourth way of using asterisk. It means multiplication. In the fourth case, integer tee is 5 times 9 and it is 45.

Now, let’s consider how to write the swap function correctly.

The swap function cannot see aee and bee. Instead, we pass their addresses to the swap function.

The arguments m and n are pointers because they have asterisks in front of them.

Calling swap uses the addresses of aee and bee by adding & in front of them.

This is what the stack memory looks like just after entering the function and before executing line cee.

The value of m is 100 because that is the address of aee. Again, letter Aee is added in front of 100 to indicate that this is an address.

The value of n is 101 because that is the address of bee.

Let’s move to the line cee. This line assigns a value to an integer u.

What is the value? The asterisk appears at the right hand size of the equal sign. Thus, we are going to use the right hand side rule. The first step is to read the value of m and it is the address of 100. The second step goes to the address 100. The third step reads the value at that address and it is 5. Thus, 5 is assigned to the local variable u.

Line dee can be decomposed into two statements by using a temporary variable t. The first statement uses the right hand side rule. First, take n’s value 101 as an address. Second, go to address 101. Third, read the value at that address. The value is 7. 7 is assigned to the temporary variable tee.

The next statement uses the left hand side rule. Take m’s value as an address. M’s value is 100. Go to address 100. Since this is the left hand side, modify the value at this address. The value of tee is 7. This statement assigns 7 to the value of address 100.

The program continues executing line ee. The asterisk is at the left hand side. The first step goes to the value of n and it is 101. The second step goes to address 101. The third step modifies the value at address 101. The value of u is 5. Thus, the value at address 101 is changed to 5.

This is what the stack memory looks like when the swap function has finished lines cee to ee.

After the swap function finished and the program resumes from line 4, this is the stack memory. Please notice that the frame for the swap function has been popped.

The value of aee is 7 and the value of bee is 5. Their values have been successfully swapped.

To summarize, the swap function needs to change two values, aee and bee. Thus, the swap function cannot use return because a function can return only one value. The swap function cannot see aee or bee directly because aee and bee are in another frame. In order to modify aee and bee, the swap function uses the addresses of aee and bee. That means the swap function uses pointers as the arguments.

Please become familiar with the four ways of using asterisk. The first is to create a pointer and a type must be used. The second is the left hand side of an assignment. The third is the right hand side of an assignment. Finally, asterisk can also be used for multiplication.