This lecture provides more details about pointers. More specifically, this lecture focuses on the syntax and their meanings.

This is a review of the previous lecture. We need to use pointers for writing the swap function. This slide shows the stack memory when the swap function has finished all statements and the top frame is about to pop. By using the addresses of aee and bee, the arguments m and n can modify the values of aee and bee even though aee and bee are not in the frame of the swap function. Please remember the rules for asterisk based on whether it is on the left hand side of assignment, or right hand side of assignment.

Please be aware that the right hand side rule may be applied even when there is no equal sign. These are two examples.

This example first creates an integer aee and assigns 2020 to the value of aee.

Then, the program creates an integer pointer called pee. Its value is the address of aee.

We can print the value stored at the address pointed by pee. Asterisk pee follows the right hand side rule: First, take pee’s value as an address. Pee’s value is 100. Second, go to that address. Thus, we go to address 100. Third, read the value at that address. The value is 2020. This print F statement will print the value 2020.

The next function call is another example. Function F takes one argument and it is an integer. When calling the function, asterisk pee is given. This also uses the right hand side rule and reads the value of aee. Hence, the value of tee in the function F is 2020.

We need to understand the meaning of data types.

What are data types? They specify what information is available and what operations are allowed.

In C language, some data types are already available, such as integer, character, and double. Programmers can create new data types, such as car, desk, phone, and light bulb.

Why are data types important? Because data types tell us what we can and cannot do.

Let’s consider a new data type called car.

A car has information about the engine size, the number of seats, the size of the fuel tank, and so on. These are the information about a car. You can drive a car. You can accelerate and decelerate.

In contrast, a desk has length, width, and height. However, a desk has no concept of engine size. You cannot accelerate or decelerate a desk. You can write on a desk.

If you have a data type of a phone, the information may include the size of the screen, the amount of storage, and so on. You can use a phone to make a call or to send a text message. Your phone may have map for navigation when you walk.

You probably do not want to carry your desk when you walk. Your desk does not help you navigate.

You cannot mix data types because you cannot send a text message using a car. Nor should you accelerate a phone.

A phone has no engine.

Do not mix data types because doing so makes no sense.

Let’s get into even more details about pointers.

Imagine that tee one and tee two are two data types.

We can create a variable x and its data type is tee one. That means the value of x stores information about the data type tee one.

If we add ampersand in front of x, then we are getting the address of x. This is the data type of tee one asterisk because it is an address.

Next, we may create a pointer called y. The value of y is an address. At this address, stores the value of type tee two.

If we add asterisk in front of y, then we either modify or read the value at that address. The value is type tee two.

Let’s see some more examples. We first create an integer aee and its value is 5. Another integer bee has value 7.

The third line creates a pointer pee. This pointer stores an address. At that address is an integer.

The fourth line assigns the address of aee to the value of pee.

Please notice that the third and the fourth lines can be combined into a single line shown in the inserted callout here. They mean exactly the same thing.

The fifth line changes the value of pee to the address of bee. Now, the value of pee is address 101.

The sixth line is an error because the types do not match. Pee is an integer pointer and aee is an integer.

The seventh’s line creates another integer pointer called Q.

.

The eighth line assigns pee’s value to the value of Q.

.

Thus, the value of Q. is address 101.

.

Let’s see even more examples.

We can use the fourth line to do two things: creating an integer pointer and also assigning the value of pee to the value of Q.

.

The fourth line can be divided into two statements shown in the callout.

.

.

This is a review of the left hand side rule. Asterisk pee takes pee’s value as an address. Pee’s value is address 101. Thus, we go to address 101. The third step assigns value -264 to this address.

This is equivalent to assigning -264 to the value of b.

The sixth line creates an integer cee. Cee’s value uses the right hand side rule by reading the value at address 101. Thus, cee’s value is -264.

The seventh line assigns the address of cee to the value of Q.

.

Now, Q’s value is address 104.

The eighth line is an error because their types do not match. Cee is an integer and Q is an integer pointer.

The ninth line is also an error. Cee is an integer, not a pointer. Thus, we cannot store the address of aee in cee.

The tenth line is an error because we, as programmers, cannot change the address of anything. Our programs have to take the addresses assigned by operating systems.

The last line of this slide is also an error. Pee is a pointer and it must stores an address. The value 2020 is an integer, not an address. Thus, this is an error.

This slide shows more examples of errors. All errors in this slide involve mixing types of integers and integer pointers.

Mixing types is a common mistake. Please understand the type rules explain in this lecture.

If you mix types, your programs usually cannot pass compilation and cannot run at all. In some cases, you can get an executable file but the program’s behavior will be wrong.

In most cases, G C C can detect the problems when you mix types. When this occurs, you must correct the errors. If you ignore what G C C tells you, your program will not work.

The following is a true story. A few years ago, a student came to the office hour of a teaching assistant.

The student said, “My program does not work. I have not slept for two days.”

The teaching assistant asked, “Do you notice this gcc warning about types?”

The student said, “Yes, I will worry about that after making my program work.”

The teaching assistant said, “This is your problem. You need to add \* in front of a pointer.”

The student said, “It works now. I spent 30 **hours** on finding this problem.”

The teaching assistant said, “It took me 30 **seconds** because gcc told me the problem.”

The student happily left the office hour.

Please understand that G C C is usually good detecting type errors. However, sometimes G C C fails to detect type errors. You must understand the types. Do not rely on G C C completely.

Let’s move to a new topic about types.

So far we assume that everything takes only one unit of memory. We use 100, 101, 102, for aee, bee, pee, and so on.

This is not correct. Different data types need different amounts of memory. You can imagine that a car is very complex, with many components. In contrast, a desk is relatively simple. Thus, the information about a car needs more memory space than the information about a desk.

This slide shows the sizes of different data types.

In C programs, you can use size of to get the size of a data type or a variable. Size of returns a long integer. Thus, inside print F we need to use al dee.

The unit of size is byte. Each byte is 8 bits. Each bit can store two values, either one or zero.

One byte is large enough to store from zero to 255, if this byte is unsigned.

If a byte is signed, it can stores between -128 and 127.

This program reports the sizes of char, int, float, and double. The values are 1, 4, 4, and 8 respectively.

The second part of the program reports the sizes of pointers of char, int, float, and double. All of them are eight.

The size of char is one. However, the size of int may depend on machines. On this particular machine, the size of int is 4 bytes. The size of a pointer also depends on machines. On this machine, the size of a pointer is 8 bytes, i.e., 64 bits. If your laptop or desktop was bought in the past few years, it is likely to have 64 bits as the size of pointers.

You need to be aware that some machines may have fewer or more bits for the size of pointers.

The previous slide uses data types as the input to size of. We can also give variables to size of.

.

In this example, size of reports the sizes of aee, bee, and cee. When we add ampersand in front of aee, bee, or cee, we get the addresses and the size is always 8 because this machine uses 64 bits for the size of pointers.

The program creates three pointers, pee aee, pee bee, and pee cee. Their sizes are all eight.

.

If we add asterisk in front of the pointers, size of reports the sizes of char, int, and double. Thus, the values are one, four, and eight.

It is extremely important to match types. This slide shows that aee is the type of int. We want pee to be a pointer storing the address of aee. Pee’s type must be int followed by asterisk.

Bee is a variable of double. If we want to use Q to store the address of bee, Q’s type must be double followed by asterisk.

If you understand that different types have difference sizes, it should be easier to understand why we must not mix data types.

Consider these examples.

Aee is an integer and has four bytes.

Pee is a pointer to char. Thus, the size of asterisk pee is only one byte.

If we assign the address of aee to the value of pee and then assign 2020 using asterisk pee, problems will occur.

Pee is a pointer to char and asterisk pee will modify only one byte. However, 2020 is too big for one byte. Thus, the program will not successfully store 2020 in aee.

The next example has bee as int and it has four bytes.

Q’s value is address of bee.

The eighth line uses the right hand side rule of using Q. This is a problem. The right hand side wants to read 8 bytes because double has eight bytes. However, bee has only four bytes. Where will the other four bytes come from? The other four bytes will read four bytes from the memory adjacent to the memory for bee. We do not know what these four bytes are and the program will likely do some strange things.

Do not mix types.