This lecture explains the Q sort function in C language and the concept of function pointers. C’s Q sort function implements the quick sort algorithm for arrays of different types.

This lecture will explain how to use the Q sort function. This lecture will not explain the algorithm of quick sort. In order to use Q sort, it is necessary to understand function pointers.

We have seen pointers for data. A pointer is a variable and its value is a memory address.

Inside a computer, every line of program also has an address. This example comes from the first homework. Inside G D B. when you set a break point, you can see these numbers zero X four zero something. These are the addresses of the starting points of these functions.

The starting point of each function has an address. Thus, we can use a pointer to store a function’s address.

In fact, we can create a type for function pointers. Consider this example. This program has two functions at the top. Each function takes two input arguments and both of them are integers. The functions return integer.

At the middle of the file, above the main function, a new type for function pointer is created. Let’s inspect this line carefully.

The line starts with type def, meaning that we are going to create a new data type.

The next word is I N T. .

This is the return type of the function.

We know that this is a function pointer because of the syntax using parentheses and \*.

Inside the parentheses, after \*, is the name of the new data type. In this example, the new data type is called F U N C T Y P E . .

After the parentheses, there is another pair of parentheses. This is the list of arguments for the function. In this example, the function takes two arguments and both of them are integers.

Inside the main function, a pointer for function is created. This pointer is called P T R . .

The pointer’s type is F U N C T Y P E . .

After creating this function pointer, we can assign a function to this pointer. The program first assigns F U N C one to P T R . .

The next line calls the function using P T R . using 3 and 5 as the arguments. The function’s return value is written to variable cee. The program will print cee equals to 8 because this function adds the two arguments and returns the result.

It appears that P T R is just a function, even though it is actually a pointer to a function.

Next, P T R points to the second function. The program calls the function by using P T R with arguments 3 and 5. The second function returns the product of the two arguments. Thus, the value of dee is 15.

Please notice that function pointer does not use heap memory. There is no malloc and no free.

After understanding function pointer, we can talk about the Q sort function in C. . This function takes four arguments. The first is an address, the second and the third are integers, the fourth is a function.

Let’s first read the document and then explain what this really means.

The Q sort function sorts an array with N M E M B. elements of size. The base argument points to the start of the array. The contents of the array are sorted in ascending order according to a comparison function pointed to by C O M P A R, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

What does this mean?

The first argument must be an address. In C, void \* means that it is an address but the type is not specified right now.

As mentioned earlier, different types have different sizes. By using void \*, this function requires a pointer but does not yet specify the size of the data type. Inside each machine, the size of a pointer is already known. The size of pointers is determined by the hardware of a computer. For example, your laptop or desktop probably has 64 bits for pointers. That’s eight bytes.

This slide explains the meanings of the first three arguments. The Q sort function sorts an array. The first argument is the address of the first element. The second argument is the number of elements. The third argument is the size of each element.

The Q sort function selects array elements by their addresses based on the following rule: the address of element of index k is the address of the first element with index 0 plus k times the size of each element. The value of k is between zero and the number of elements.

The Q sort function takes the addresses of two elements in the array, sends the these addresses to the compare function, based on the comparison function’s result, Q sort may swap the values stored at these two addresses.

Let’s now consider the fourth argument of the Q sort function.

How do we know this argument is a function? Its type is marked by two pairs of parentheses and \* after the first open parenthesis.

What is the comparison function? This function returns an integer and takes two addresses. We know the two arguments are addresses because their types are void \*.

Putting C O N S T. in front of the pointer means this function must not use the left hand side rule to modify the value stored at the two addresses.

The document says that the comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined. In other words, when the two members are equal, the Q sort function may or may not swap the two values.

This slide shows how the Q sort function works. It selects the addresses of two elements in the array. The two addresses are marked ei one and ei two. These two addresses are sent to the comparison function.

The comparison function takes these two addresses and returns an integer that can be less than zero, equal to zero, or greater than zero. If the value is less than zero, the values at address ei one and address ei two are unchanged because vee one is already smaller than vee two. If the comparison function returns an integer greater than zero, then the value vee one is greater than vee two. The two values should be swapped. If the comparison function returns zero, Q sort may or may not swap these two values.

C provides the Q sort function. However, C does not provide the comparison function because the Q sort function can sort arrays of any type. Programmers have to provide the comparison function. A comparison function should have three steps.

The first step casts the two arguments into the right type. Please remember that these arguments are addresses. This step is necessary because different types have different sizes. If an address is a pointer to an integer, then four bytes will be read from that address. If an address is a pointer to a double, then eight bytes will be read from that address.

This example shows how to write a comparison function for an array of integers.

The first step casts the arguments to pointers of integers. Please notice that we need to have C O N S T because that is required by the Q sort function.

The second step uses the right hand side rule to read the values at the addresses.

The third step compares the two values and returns a negative integer, zero, or a positive integer.

The next comparison function will compare two vector objects by their x values. We will follow the same three step approach for writing the comparison function.

The first step is to cast the arguments into the right type. If the array elements are vectors, then the first step casts the arguments into addresses of vectors.

The second step reads the values at the addresses. In this example, we want to compare the vectors by their x values, thus, the values are P T R one arrow x and P T R two arrow x.

The third step compares the two values.

The comparison function for strings is a little more complex because each string is an array of characters. Since an array is also a pointer, the input arguments are pointers of pointers.

For the first step, let’s imagine that C has the type called string. A R G. one and A R G. two are pointers of strings. Thus, their types are string \*.

C does not have the string type. Instead, in C, a string is an array of characters. An array is a pointer. Thus, a string’s type is C H A R \*.

If we put these together, the type of A R G. one and A R G. two is char \* \*.

We also need to add C O N S T for the pointers P T R one and P T R two.

I will explain their meanings in the next slide.

The second step uses the right hand side rule to get the strings. The third step uses the S T R C M P. function to compare these two strings.

The word C O N S T has different meanings depending where we put it.

In this example, C H P T R one has C O N S T before char \*.

C H P T R two has C O N S T after char \*.

What is the difference?

When C O N S T is in front of C H A R \*, this pointer, C H P T R one cannot use the left hand side rule. However, C H P T R one itself is not a constant. Thus, we can change the value of C H P T R one and make it point to another address. As you can see, it is OK to make C H P T R one the address of the first element of S T R two.

If we put C O N S T after char \* and only in front of C H P T R two, that makes C H P T R two a constant. That means we cannot change the values of C H P T R two. However, we can use the left hand side rule to change the value pointed by C H P T R two.

We can also put C O N S T in both places, as shown here for C H P T R three. We cannot change the value of this pointer, nor can we use the left hand side rule to change the value at the address C H P T R three points to.

Let’s go back to the comparison function for strings. The two C O N S T for P T R one means that we cannot change the value of P T R one, nor can we use the left hand side rule to change the value at the address pointed by P T R one.

When writing a comparison function for Q sort, it is important to follow these three steps. The first step casts the input arguments to the correct types. The second step uses the right hand side rule to retrieve the values at the addresses. The third step compares the values and returns a negative integer, zero, or a positive integer.

We will talk about the quick sort algorithm in a later lecture. The quick sort algorithm uses a principle that is not used in selection sort or bubble sort. This principle is called transitivity. Transitivity says that if ei is greater than bee and bee is greater than cee, than ei must be greater than cee. It is unnecessary comparing ei and cee. This principle allows quick sort to avoid unnecessary comparisons. Thus, quick sort is quick.