This lecture talks about data types created by programmers.

The C language allows programmers to create data types by using structures. The word in C is S T R U C T.

.

You can find sample programs in this git hub repository.

Let’s first see some examples of struct before we explain why it is necessary creating new data types.

The first example is a new type called Vector and it has three attributes, x, y, and z.

In this example, all attributes are integers. Please notice the syntax. At the top, we need to say type define struct, followed by open brace.

The attributes are listed between the open brace and the close brace.

After the close brace, you can give a name of the data type. In this example, the name is vector. From now on, Vector is a new data type.

Please remember adding semicolon after the type. If you forget this semicolon, G C C. may get confused and produce dozens of error messages. These error messages can be solve by a single semicolon.

Usually, a new data type has its own header file. This vector structure has a header file called vector dot h.

.

In the second example on the right side, we can see that the attributes may have different data types.

The first three attributes x, y, and z are integers. The next attribute tee is double. The last attribute name is an array of 30 characters.

Why do programmers want to create new data types? The most important reason is to organize information better. Relevant information should be put together. Consider a three dimensional vector, it is better to put the three directions x, y, and z together. If you are writing a program related to transportation, you may want to have a data type called Car. Inside this Car data type, there are attributes such as the engine size, the size of fuel tank, the number of horsepower, the number of seats, and the fuel efficiency. If you are writing a program about buildings, you may want to have a data type called House. Each house has attributes such as the number of bed rooms, the number of bathrooms, the size of living rooms, whether the house has a basement, etc.

We need to distinguish the concept of data types and the instances of these.

Data types are sometimes called abstract data types because data types are different from concrete examples. Car is a data type and it does not refer to any specific car. A car is a concept.

Your car is an instance of the Car type. We cannot say the Car type has five seats or the fuel tank is 15 gallons. Instead, when we say a car has five seats, we refer to a specific car, i.e., an instance of Car.

Similarly, an integer is a data type. We cannot say integer is 5. Instead, we can create an instance of integer and assign 5 to this particular integer.

C is not an object-oriented programming language. In an object-oriented language, a class is a data type, similar to the concept of a structure. An instance of a class is called an object.

In other words, an object is an instantiation of a class.

We will borrow the terminology from object-oriented languages. We will use the word object to refer to an instance of a data type.

By creating new data types and organizing relevant information together, it is easier to write complex programs without mistakes. Imagine that we cannot create a data type called Car. Then, we have to remember to keep track of the information about cars, such as sizes of engines and sizes of fuel tanks. Without a structure, it is very easy to forget putting these attributes together and programs will have mistakes.

Another advantages of a structure is that the attributes can be passed to a function as an argument all at once. Think of about the second example of the Vector structure. The structure has five attributes. It is possible passing all attributes together as a single argument to a function. Without the structure, the five attributes have to become five arguments.

Another reason is to maintain consistency. Let’s imagine that you create a data type called Address. When someone moves, the person’s address needs to change the street number, the street name, maybe the city and the zip code. When the information about address is put together, programmers are more likely to remember and the data can be maintained properly.

Some object-oriented programming languages, such as Java and C plus plus, use classes instead of structures. It is possible to protect the attributes of classes from being accidentally changed. C is not object-oriented so this feature of data protection is not available.

Let’s see how the Vector structure can be used. This is an example creating a vector object called vee one.

This vee one object will reside in the stack memory. We can assign values to the attributes using vee one dot x, vee one dot y, or vee one dot z.

What is the address of vee one dot x? The address is determined by the G C C compiler and the operating system together. In this example, the address of vee one dot x is 300.

After the address of vee one dot x is decided, the address of vee one dot y is the address of vee one dot x plus the size of vee one dot x. Since x is an integer, it uses 4 bytes. Thus, the address of vee one dot y is 304. The address of vee dot z is the address of vee one dot x plus the size of x and the size of y. Since both x and y are integers, the address of vee one dot z is 308.

This slide shows another example using the Vector structure. Vee one is the same as the previous example. This example adds another object called vee two.

This is the output of the program. I will explain the program in more details.

It is possible initializing all attributes of vee two to zero using this method: brace with only one zero inside.

It is also possible assigning the values of vee one’s attributes to vee two’s attributes.

Please notice that assignment is the only allowed operators for structures. C does not allow operator overloading.

C does not allows any other operators for structures such as not equal, less than, less than or equal, greater than, greater than or equal, increment or decrement.

After the assignment, vee one’s x, y, and z and vee two’s x, y, and z have the same values. However, they occupy different memory space. Thus, changing vee one dot x does not change vee two dot x. Changing vee two dot y does not change vee one dot y.

Let’s take an even closer look of the program by understanding how the stack memory changes.

When we create the vee one object, the space for three integers, vee one dot x, vee one dot y, and vee one dot z are put on the stack memory. Since the object has not been initialized, the values are marked u, meaning unknown.

The next three lines assign values to the three attributes, x, y, and z.

Next, we create a Vector object called vee two. All attributes are initialized to zero. Please notice that vee two occupies different memory space in the stack memory from the space for vee one.

C allows using the equal sign to assign the attributes’ values from vee one to vee two.

Because vee one and vee two use different memory space, modifying vee one’s x does not affect vee two’s x. Modifying vee two’s y does not affect vee one’s y.

The next example shows passing an object to functions. This is the program’s output.

The program has two functions called print vector and change vector. In the main function, the vector’s attributes are 3, 6, and -2. This vector is passed to the change vector function as an argument. The vector’s attributes are changed to 5, -3, and 7. After leaving the change vector function and back to the main function, the vector’s attributes are still 3, 6, and -2. What is happening? What does this mean?

When passing the vector object vee one from main to the change vector function, a new object is created for the argument. This object is called vee and it is on the stack of the called function. The attributes are copied from the main function to the change vector attribute by attribute. Because the argument vee occupies different memory space from the memory space for vee one, changing vee inside the function has no effect on the vee one object inside the main function.

If you are familiar with Java, Python, or C plus plus, this argument passing in C is different.

In Java or Python, passing an object means passing by reference. Thus, the changes inside a function will be kept in the calling function when the called function finishes.

In C plus plus, you can choose to pass by reference. In that case, C plus plus is the same as Java or Python.

C does not have the concept of passing by reference. Calling a function using an object means creating a new object on the stack memory.

What should we do if we want to change the vector inside the main function? We will use the same method as before: passing the address.

In this example, the main function calls the change vector function using the address of vee one. The function’s argument must be a pointer.

To get the address of vee one, add ampersand in front of vee one when calling the function.

Inside the function, we use vee arrow to refer to x, y, or z.

The arrow is written by using the minus sign followed by the greater than sign.

Please notice that inside the change vector function, vee is a pointer and stores the address of vee one’s dot x.

This slide shows another way to initialize the attributes. Vee two is a local object. The attributes can be assigned by using dot x, dot y, and dot z. We can use the left hand side rule of vee. \* vee equals vee two will copy the values of vee two’s attributes to the memory pointed by vee.

The print vector function uses the right hand side rule of vee. The attributes are read from the memory address pointed by vee and copied to the argument vee of print vector.

When should we use dot and when should we use arrow?

The rule is simple: if it is a pointer, use arrow. If it is not a pointer, use dot.

Consider this example. An object v is created. Since it is not a pointer, we use vee dot x.

A pointer called vee pee is created and its value is the address of vee. Because vee pee is a pointer, we need to use arrow.

We can also have a structure using another structure. This example creates a structure called date of birth. This structure has three attributes for year, month, and date.

Another structure called Person contains two attributes. The first is a pointer for name. The second is the date of birth structure. From this example, we can see that it is possible creating a structure using another structure.