This lecture talks about a new topic: linked list.

Before we get into the details of linked list, let’s review what we have learned about memory management. We have learned two ways to manage memory. The first is to allocate memory when a program is written. The data will be stored in the stack memory. This method is very restrictive because the program cannot process data larger than the program anticipates.

The second method decides the amount of memory needed after a program starts. For example, a program needs to read a file and stores the data in the file. The program may open the file, read the entire file and decide the amount of data in the file, allocate the memory, and the read the file again. Before the program ends, the program free the memory.

Even though the second method is more flexible than the first method, the second method is still too restrictive.

The third method is even more flexible. This method allocates memory as needed and free when the memory is no longer needed. This is called dynamic structures. This is widely used for solving many problems, such as database, web users, text editors, and many others.

When you create a database, you do not know how many pieces of data may be stored in the database. You cannot allocate a very large amount of memory because that is wasteful. If the allocated memory is too small, that becomes a problem.

When you create a website, you do not know how many users may register. Your website should allocate memory as more users register.

Another example is a text editor. If you write a program as a text editor, you do not know how many pages a user may write.

Thus, a good solution is to allocate memory as needed.

This slide shows a general concept about how dynamic structures work.

A pointer in the stack memory stores the address of a piece of heap memory. Each piece of memory contains two parts: The first part is a pointer to another piece of memory. The second part stores the data.

Let’s inspect this carefully. Suppose P T R. is a pointer in the stack memory and its value is 20000. This address is in the heap memory. At address 20000 is another pointer and its value stores the address 50000. Meanwhile, data is stored in this piece of memory. Through the pointer, we can find another piece of memory at address 50000. This piece of memory has a pointer storing the address 40000. This piece of memory also contains some data.

We can follow this pointer and go to address 40000. This piece of memory has a pointer and it stores the address 30000. This piece of memory also contains data. Through this pointer, we can reach the address 30000. At the address 30000, the pointer stores NULL. NULL is an invalid memory address. NULL is used to indicate that no more data is stored after this piece of memory.

At the lower left of this slide, we can express this dynamic structure as a linear structure. This structure is called linked list.

The pointer P T R. points to address 20000. From 20000, we can go to address 50000, then 40000, then 30000. This dynamic structure ends when reaching the special character NULL.

Each piece of memory is called a node.

Here are some frequently asked questions about dynamic structures.

First, why do we use heap memory? Heap memory can be allocated when more memory is needed. When the memory is no longer needed, the memory can be freed. Stack memory does not have this property.

We will need local variables or arguments to store the address of the first piece of memory in the linked list.

Another advantage of heap memory is that heap memory can be accessed by different functions.

Why are the addresses scattered? Why are the memory addresses of the list node not increasing or decreasing? First, malloc returns the address of available memory. Malloc does not necessarily return increasing or decreasing addresses.

Another reason why the addresses are scattered is that the allocated memory may be freed. After malloc and free several times, the memory will likely be scattered.

A linked list is an example of a container structure. What is a container structure? Each piece of memory may store some data. The structure is the same for different types of data. Each node has a pointer to another piece of memory. Because this structure can store different types of data , the structure is like a container. Thus, we call this a container structure. If each piece of memory has only one pointer, we call this structure a linked list.

It is certainly possible that each piece of memory has two pointers.

In this slide, each piece of heap memory has two pointers. A pointer called P T R. in the stack memory points to heap memory address 20000. The piece of memory at 20000 has two pointers storing the addresses of 40000 and 50000. We can draw the memory in a two dimensional way. Consider the node at address 20000. The address 40000 is at the left side of the node. The address 50000 is at the right side of the node.

The piece of memory at 40000 has two pointers storing the addresses of 60000 and NULL. NULL is invalid for memory address. This means the piece of memory at 40000 points to only one valid address of 60000.

The piece of memory at 60000 has two pointers and both store NULL. That means this piece of memory does not point to anything further.

Next, let’s consider the piece of memory at 50000. It has two pointers storing NULL and the addresses of 30000.

The piece of memory at 30000 has two pointers storing the addresses of NULL and 70000.

When each piece of memory has two links, we call it a binary tree.

Usually, we do not draw the links pointing to NULL. As you can see in this slide, they become gray.

A common question from many students is the difference between linked list and binary tree. The difference is profound due to the important properties of number theory.

In a linked list, each piece of memory has only one link. In a binary tree, each piece of memory has two links.

One and two a fundamentally different. This is the reason. For any positive number N, it is possible finding a number K. such that two of power K. is N. .

For example, if N is 4, K is 2. If N is 8, K is 3. If N is 0.5, K is -1. .

This same property also applies to 3. For any positive number N, it is possible finding a number M. such that three of power M. is N. .

For example, if N is 9, M is 2. If N is 27, M is 3. If N is one third, M is -1. .

Therefore, two and three can do the same thing: they can be used as the bases to express different numbers.

One is different. Any power of one is still one. Thus, one is limited. We cannot use one to express another number.

Let’s examine the linked list and the binary tree again using a different way to see them.

A linked list is one dimensional. If we want to move to the middle of a linked list, we have to pass one node at a time. In other words, we have to visit half of the nodes before we can reach the middle of a linked list.

A binary tree is two dimensional. It is possible arranging the data so that about half data is on one side and the other half on the other side. As a result, it is possible to eliminate half of the data in a single step by deciding to go to the left side or the right side.

After we understand the concept of linked list and binary tree. Let’s into more details about linked list.

There are different ways to create linked lists. This is one way to do it. A linked list is composed of one or many nodes. Each node is a structure. This uses typedef to create a new type. This data type is a structure and it has a temporary name called list node.

The first attribute is a pointer of the type struct list node. It is important that they have the same name. Moreover, this must be a pointer for two reasons. First, we need a pointer so that we can use heap memory. Second, this must be a pointer because the size of a pointer is the size of the address and this is fixed for each machine. Since we have not finished the definition of the structure, the size of this structure is not known yet. If this is not a pointer, G C C. does not know how much space is needed for the structure.

At the end of this new type, we add a name called node. This is a new data type. From now on, we can use node as a data type.

A node may include many different types of data. In this example, the data includes an integer, an array of 20 characters, and a double.

In this example, the pointer is the first attribute of the structure but it does not have to be the first attribute. It can be move to later in the structure and makes no difference.

As mentioned earlier, a linked list or a binary tree is a container structure. A container structure usually supports at least four types of operations: insert data, delete data, search whether a piece of data is stored, and destroy.

Destroy deletes everything.

Let’s use a simple linked list. This structure has only two attributes: a pointer and an integer.

In the rest of this lecture, we will study three functions. The first one is a static function called node construct. This function is called by the list insert function only. This function is static and is not visible outside this particular file. A static function can be called by only functions in the same file.

This node construct function takes one integer as the argument. Inside this construct function, it allocates memory for a node. Please notice that node is a type we defined earlier. Each node has two attributes. The construct function assigns the value from the argument to the attribute value. The attribute next is initialize to NULL.

It is extremely important that you initialize the next pointer to NULL. If you forget, your program will be wrong. This is a very common mistake. Every semester I have to remind students many times and this mistake still occurs. I hope nobody makes this mistake this semester. The construct function returns the memory by malloc.

Next, let’s consider the insert function. The function takes two arguments. The first is a pointer for the node. The second is an integer. The first line prints the integer. The second line calls the construct function. The third line assigns the first argument to the next attribute of P. . The function returns P at the end.

We will explain how these two functions work for creating a linked list.

This slide shows how to call the insert function. We first create a pointer of Node and initialize it to NULL. It is really important to initialize this pointer to NULL. If you do not initialize it, your program will be wrong.

Then, the program calls the insert function twice for inserting three numbers, 917, negative 504, and 326 into the linked list.

This slide shows the stack memory before calling the insert function.

Let’s start from the first line of the main function. It creates a pointer of type Node and sets the value to NULL. This is a local variable so it is stored in the stack memory.

Next, the main function calls the insert function. As always, if there is a function call, a new frame is pushed to the stack memory. The argument h stores the value NULL because it is the value of head in the main function. The second argument is 917. The value of P is unknown because it has not been initialized yet. The value address is the address of head in the main function.

The insert function calls the construct function. This slide puts both functions together.

The construct function has only argument and its value is 917. The local variable n’s value is unknown.

At the top of the construct function, it calls malloc. A piece of heap memory at address 10000 is allocated and this piece of memory has two attributes for next and values. Since neither has been initialized, both of them are marked U as unknown for now.

The next two lines assign values to the attributes. First, 917 is assigned to the value of this structure.

The next line assigns NULL to the next attribute.

The next line returns N. . We need to go to the stack memory and check the value address. The value of N is assigned to the value of P. .

Then, the top frame for the construct function is popped.

The next line assign h to P’s next. P’s value is 10000. P’s next is at the address 10000. H’s value is NULL. Thus, this line assigns NULL to the value at address 10000.

Right now, it seems that this step is unnecessary because the value at address 10000 is already NULL.

Later, we will see that this step is important when connecting the second node in the linked list.

Finally, the insert function returns P. . This value is written to the value at address 200. The variable head in the main function stores address 10000.

After the insert function terminates, the top frame is popped and this is the stack and heap memory.

Next, the main function calls the insert function again. This time, we will see how a new node is inserted into a linked list that already has one node.

Another frame is pushed to the top of the stack memory. This time, the value of H is address 10000. The value of V is negative 504.

The insert function calls the construct function. The construct function allocates heap memory. The value of next is NULL. Negative 504 is stored in the structure’s value.

The next line connects the two nodes of the linked list. This line assigns H’s value to P’s next. P’s value is address 20000. This line changes the value stored at address 20000 to address 10000. Thus, the node at address 10000 is reachable from the node at address 20000.

To put it another way, the node at address 20000 is in front of the node at address 10000.

The next line return P’s value and stores it in the head in the main function. The head’s value now becomes 20000.

This slides draws the linked list. Head’s stores the value of address 20000. The structure at this address stores an integer of value negative 504. The structure’s next stores address 10000. The structure at this address stores an integer of value 917. The structure’s next stores NULL. Thus, there is no more node.

The program can continue inserting another value of 326. This new value will be stored in a node as the first node of the linked list.

An important step of the linked list is this line. When the insert function is called, H points to the first node of the linked list. This line make the node pointed by P the node before the original head. Thus, the new node becomes the first node of the new linked list.