This lecture continues our discussion about structures. In particular, we will talk about structures with pointers and heap memory.

Let’s use the Vector structure introduced last time as an example. This vector structure has three attributes and all of them are integers.

On the left side of this slide, we can see the definition of the structure.

On the right side, we can see how to create a vector object on the heap memory. The first function is called vector construct. The concept of constructor comes from object oriented programming. A constructor creates an object and initialize all attributes. It is good to use constructors so that all attributes are properly initialized.

The concept of constructors is very important when handling structures. A complex structure may have many attributes. Moreover, over time, some attributes may be added and some attributes may be removed.

By using constructors, if more attributes are added or some attributes are removed, we need to change only one place. This can dramatically reduce the chances of mistakes.

The destructor is a similar concept. It is a place where heap memory is freed. Please notice the correspondence. The constructor calls malloc once. The destructor calls free once.

This slide shows how to use the constructor and the destructor. In the main function, it creates a pointer called vee one. The constructor returns the allocated memory and assigns the address to Vee one. The constructor also assigns values to the attributes.

The print function can print the attributes of Vee one.

Before the program ends, the main function calls the destructor to free the memory.

Let’s take a closer look and understand what happens to the stack and heap memory when the main function calls the constructor.

In the main function, a pointer vee one is created. This is a local variable; thus, it is stored in the stack memory. Its value is unknown.

Next, the main function calls vector construct. This is a function call. A new frame is pushed to the top of the stack. The return location is the line after the function call. The value address is the address of vee one. In this example, vee one’s address is 100 and the value address is 100.

The three arguments ei, bee, and cee are 3, 6, and -2 respectively.

Let’s see what happens inside the construct function. The function has a local variable called vee. It is stored in the stack.

At this moment, the value of vee is unknown and is marked U in the stack memory.

The next line uses malloc to allocate memory in the heap memory. Since a vector object has three integers, 12 bytes will be allocated in the heap memory. Suppose the starting address is 10000. The address is stored in the value of vee.

The values in the heap memory is not initialized yet. Thus, the values at address 10000, 10004, and 10008 are all marked U, meaning unknown.

Since vee is a pointer, we need to use arrows for the attributes x, y, and z. The values are assigned to 3, 6, and -2 respectively.

Returning vee means returning the value of vee. The value is address 10000. The value address of this function call is address 100. Thus, 10000 is written to the value of address 100.

After the constructor finishes and the program resumes at the main function, the value of vee one is 10000.

Before the program ends, the destructor is called to free the memory. The constructor frees the 12 bytes allocated earlier by malloc.

It is important to understand that the value of vee one is not changed by the destructor. The value of vee one is still 10000. It is the programmers’ responsibility not to use vee one any more in the rest of this program.

This slide explains the situation more clearly. Suppose pee is a pointer of Vector. The program uses malloc to allocate memory for one vector object.

Calling free pee releases the heap memory pointed by pee. Pee’s value is unchanged. We know pee’s value is unchanged because we use pee, not ampersand pee, when calling free pee.

Pee’s value is still the heap memory allocated earlier when calling malloc. Thus, if we have a condition comparing pee with NULL, this condition will be false.

After free pee, the heap memory is no longer valid. Thus, pee arrow x will cause segmentation fault.

The free function cannot change pee’s value because the function is called by using pee, not ampersand pee.

Can we add ampersand in front of pee so that the free function will set pee’s value to NULL?

No. This will not work.

The free function takes an address and release the memory at that address.

If we add ampersand in front of pee, we will get pee’s address. However, pee is a local variable. That means pee resides on the stack memory. Ampersand pee is an address in the stack memory.

It is not possible to free stack memory. Thus, free ampersand pee is wrong.

What does segmentation fault mean?

Inside a computer, memory is divided into smaller units called segments. The operating system gives each program several segments to store data. When a program X intends to access data in a segment that belongs to another program Y, the operating system will stop the program X. .

This is to protect the space of program Y. .

 In other words, segmentation fault occurs when a program intends to violate the computer’s security protection.

Let me explain this by using an analogy. A computer is like a hotel. Each program is like a guest in the hotel. The hotel assigns guests to different rooms. Each room is a segment. If a guest X intends to enter the room that has been assigned to another guest Y, guest X violates the hotel’s rules.

To prevent segmentation faults, your programs should malloc memory before using the space. The programs must not use memory after free. Also, the programs must not call free twice. Remember, one malloc corresponds to one free, always.

The next topic is about structures that have pointers as attributes.

Consider this structure called person. The structure has four attributes: three integers for the year, month, and date of this person’s date of birth. The last attribute is a pointer to character for this person’s name.

The header file has some functions related to this structure. The most important function for now is the constructor. The constructor will create a person object in the heap memory. This function returns a pointer to a person object. The function takes four arguments for the name, the year, the month, and the date of this person’s date of birth.

This function first create a local variable as a pointer of person. Then, the function allocates memory on the heap memory. The person’s name needs to have space before S T R C P Y. .

Please remember to add one for the ending null character.

The other attributes for year, month, and date can be assigned. Then, the constructor returns the value of the pointer.

This slide shows how the constructor can be used. The main function has three pointers pee one, pee two, and pee three. Pee one and pee two store the addresses of the heap memory from the constructor. Pee three copies the attributes from pee one’s attributes. As you can see, the copy function calls the constructor.

The destructor frees the memory. Please notice that the destructor frees memory in the reverse order in the constructor. The construct malloc memory for pee first and then pee arrow name. The destructor frees the memory of pee arrow name first and then free the memory of pee. It is important to understand that the order must not change.

We must allocate memory for pee before allocating memory for pee arrow name. The reason is that pee arrow name does not exist before malloc pee.

In contrast, in the destructor, we must free pee arrow name before free pee. If we free pee first, then pee arrow name does not exist any more and cannot be freed.

The next topic is shallow copy or deep copy.

The problem occurs when a structure has one or several attributes that are pointers.

Let’s understand what may happen. Suppose the main function calls the constructor to create an object and assigns the result to a local variable called pee one.

This shows the stack memory immediately after entering the constructor. The return location is the line after calling the constructor. The value address is the address of pee one.

Please notice that the argument n is a pointer. Its value stores the address of the string A M Y followed by the null character. The other arguments are integers, with values 1989, 8, and 21.

When the constructor reaches the first line, a local variable pee is created. This is a pointer and its value is unknown.

The next line allocates memory by calling malloc. Suppose malloc returns the space in the heap memory and 10000 is the starting address of this piece of memory. The value of pee is 10000.

The next line calls malloc again. Suppose the malloc returns address 25000. This address is written to the value of pee arrow name.

Please notice that calling malloc twice does not necessarily get memory addresses that are adjacent. It is possible that there is a gap between the two memory addresses.

The next line copies the string from the stack memory to the heap memory.

The next three lines assign the values to the heap memory.

At the end of the function, the value of pee is returned. The value address is 100. Thus, the address 10000 is written to the value at address 100.

After the function returns, the frame for the constructor is popped. Please notice that the space allocated in the heap memory is still there, until the program calls free to release the memory.

What will happen if the program calls the copy function? As we have seen earlier, the copy function calls the constructor and the constructor allocates memory.

Thus, pee three will point to a different pieces of memory from where pee one points to.

In contrast, if we create a pointer called pee four without using the constructor, pee four’s value is the same as pee one’s value. In this case, the two pointers point to the same memory address.

Please notice that changing pee one arrow year will also change pee four arrow year.

What will happen if we use the copy function for pee three first, assign pee one’s value to pee four, and then assign pee four’s value to pee three?

This slide shows the memory after finishing the second statement. Pee four’s value is the same as pee one’s value. Pee three’s value is different, meaning that pee three points to another piece of memory.

After the third statement, the value of pee four is assigned to the value of pee three. In this case, pee one, pee three, and pee four all point to the same memory address.

Meanwhile, no pointer has value 20000. That means this piece of memory is no longer reachable. This piece of memory is lost. In other words, memory leak occurs.

Next, consider a different scenario.

This program creates a person object called pee one. Please notice that there is no \* in front of pee one. Thus, pee one is an object, not a pointer.

We can assign values to the attributes. Because pee one is an object, not a pointer, we need to use dot, not arrow.

Pee one dot name is still a pointer. Thus, we have to allocate memory for it. We can use the S T R D U P function introduced in an earlier lecture.

The S T R D U P function will allocate memory and copy the string. Suppose this piece of heap memory starts at address 25000. This is what the memory looks like now.

What will happen if we create another person object called pee two? Please notice that there is no \* in front of pee two. Thus, pee two is an object, not a pointer.

Pee two is an object and the stack memory has pee two’s attributes. Assigning pee one to pee two means copying the values of pee one’s attributes to pee two’s attributes.

For year, month, and date, they are all right because they are integers and they have space in the stack memory.

For the name attribute, it is a pointer. Copying pee one dot name to pee two dot name means that the two pointers now point to the same piece of heap memory.

This can be understood in another way. Only one piece of heap memory is allocated by S T R D U P. .

Since S T R D U P is called only once and malloc is not called, only one piece of heap memory is allocated. The two pointers have the same value and point to the same heap memory.

Since the two pointers have the same value, changing pee one’s name will also change pee two’s name.

What is shallow copy? What is deep copy? If a structure has a pointer, you need to be very careful. Assignment will copy one object’s attributes to another object’s attributes. If one of the attributes is a pointer, then the two pointers will have same value. Hence, the two pointers refer to the same memory. This is called shallow copy.

When you use shallow copy, changing one object also changes the other object. For example, if you change the first element of pee two dot name, you also change the first element of pee one dot name.

Deep copy means a program allocate memory for different pointers. As a result, the two pointers have different values. Changing one does not affect the other.

Why do we have shallow copy? Why do we have deep copy? Is one better than the other?

The answer is that both can be useful. You need to understand the advantages and disadvantages.

In shallow copy, multiple pointers point to the same heap memory. In deep copy, they point to different pieces of heap memory.

Shallow copy can save memory space but changing one will also change all the others. In deep copy, changing one will not affect the others.

Sometimes, sharing is desired. Imagine that you are building the database for a company. One of the attribute is the company’s address. Since everyone works for this company, they all share the same address. If the company moves, every employee’s address is also moved together. In this case, sharing makes sense.

Next, consider another scenario. A family has a new child. In the database for family members, this child’s address is the same as the parents’ address. When this child grows up, this person may want to move to another place and no longer lives with the parents. In this case, deep copy makes sense because doing so allows the child to have the same address at the beginning and a different address later.

Thus, both shallow copy and deep copy can be useful. You need to understand both and choose based on your needs.

Another option is called copy on write. When the values are the same, only one piece of memory is allocated and it is shared. When modification is needed, additional memory is allocated. This is an advance topic beyond of the scope of this class.