This lecture continues the topic about heap memory.

Please remember that if an array has n elements, valid indexes are 0, 1, 2, up to n minus one.

N is an invalid index.

Let me show you an example what may happen if you use a wrong index.

Consider this example. The program has local variables aee, bee, cee, x, y, and z. Their values are 1, 2, 3, 4, 5, and 6.

This program has an array of five elements. The elements are 0, -1, -2, -3, and -4. This program has a problem because the index actually goes to 5. Thus, this program is wrong.

What may happen?

At the first glance, the program appears all right. When we print the values of the array, the program prints 0, -1, -2, -3, and -4. The program even prints -5 even though 5 is not a valid index.

When we print the values of aee, bee, cee, they are correct: 1, 2, and 3.

When we print the values of x and y, their values are also correct: 4 and 5.

When we print the value of z, however, its value should be 6 but the value has been changed to -5.

The incorrect array index changes the value of z.

This example shows that the damage of using a wrong array index may not be so obvious at the beginning. Thus, it is necessary to be very careful about array indexes.

Next, let’s see how heap memory is used in the second and the third programming assignments. Both assignments manage memory for you. Thus, you can use arrays as they are given. Now is the time to understand what these programs have done.

In the second programming assignment, we need to read the content of a file to determine the amount of data in the input file. This is done in lines 39 to 42. After counting the number of integers in the file, line 45 uses malloc to allocate memory.

This is an example when the amount of memory is not known while writing the program. The amount of memory is known only after given a particular file.

This program checks whether malloc succeeds or fails. At line 46, if malloc fails, malloc returns NULL. NULL is a special symbol indicating invalid memory address.

Why does malloc fail? Two reasons. First, if the amount of memory to be allocated is too large, malloc may fail. The second reason is when a program leaks memory. Memory leak will shrink the amount of memory available to the program. If a program leaks memory, eventually the program will run out of memory.

Do you remember that malloc and free always come in pairs? The program uses free to release memory at line 75.

In the third programming assignment, memory is allocated using malloc at line 15. The memory is released at line 41.

Do you remember what type needs to be used inside size of?

A R R is a pointer of integer. In other words, A R R’s type is I N T \*. Thus, we should use I N T inside size of.

The type rule says that when an asterisk is added in front of A R R, the type becomes I N T.

.

Next, let’s understand what happens when a piece of heap memory is passed to a function. This is the program.

This is the output when we run the program.

Let’s start from the main function. Line 35 calls malloc to allocate memory. This line allocates memory for five integers.

As explained earlier, it is a good habit to have malloc and free inside the same function so that we know they appear together. Line 46 calls free to release the memory.

Lines 37 to 40 assigns 0, 1, 2, 3, and 4 to the array’s elements.

Lines 41 to 45 calls several functions.

Let’s take a closer look how the array is used inside the functions. As you can see, the array is used in the same way.

To really understand what is happening, we have to study the stack and heap memory.

This slide shows the status of the stack and the heap memory when the program reaches line 34. At this moment, the integer size has been assigned the value of 5. The pointer A R R has not been initialized. Thus, its value is marked U as unknown.

Please remember, if a pointer is not initialized, its value is unknown. It is not necessarily zero. It may be zero. It may be not.

Line 35 uses malloc to allocate memory.

Malloc finds a piece of memory in heap and returns the starting address of this piece of memory.

Suppose malloc returns 2000, the value of A R R is 2000. Please notice that the allocated heap memory is not initialized yet. Thus, their values are marked U.

.

Line 36 creates an integer called eye. Its value is not initialized. It is put on the stack memory.

Lines 37 to 40 assign 0, 1, 2, 3, and 4 to the array’s elements.

Line 41 calls a function using A R R and size as the arguments. How does this work?

This slide shows the changes in the stack memory. Calling print A R R is a function call. Thus, a new frame is pushed to the top of the stack memory. This frame has return location as line 42.

The arguments are called A R R and size but they are not the same A R R and size in the main function. They occupy different memory locations in the stack memory. When calling the print A R R function, the value of A R R in the main function is copied to the value of the A R R in the argument. Thus, both of them have the same value as address 2000. Please notice the type of A R R in the argument. The argument’s type is I N T \*. Thus, we know A R R is a pointer.

Similarly, the value of size in the main function is copied to the value of size in the argument. Both of them have value 5.

Inside the print A R R function, the array’s values are read.

What does A R R with index eye mean?

It means taking A R R as an address. We know that A R R’s value is an address because A R R’s type is a pointer.

The next step takes the value of eye and multiplies it with the size of each element. A R R is a pointer to an integer. Thus, we know the size of one element is 4 bytes. This step multiplies eye with four and add the result to the address pointed by A R R. .

The added value is treated as an address.

The fourth step goes to that address.

The next step depends on whether it is at the left hand side of assignment or the right hand side.

If it is the left hand side, the value at the address is modified. If it is the right hand side, this step reads the value at that address.

In this example, A R R eye is used in print F. . Thus, it is equivalent to the right hand side rule and the value is read from that address.

This is how the function print A R R works.

The other two functions, double A R R and triple A R R, follow the same rules.

This equation explains how addresses of array elements are calculated. The address of the element with index k is the address of the first element (index is 0) + k multiplied with the size of one element.

When a piece of heap memory is no longer needed, the memory needs to be released by calling free. If a program does not call free, the program leaks memory.

Fortunately, we can use a tool called val grind to detect memory leak.

To use val grind, we will create an alias so that all the functions are enabled. This command says that we want to use the memory check function. The result will be saved in a file called V A R L O G. . We want val grind to check memory leak. The verbose mode gives us more information.

Then, we put line 46 into a comment. This program does not release memory.

To use val grind, add val grind in front of the program. The program’s output seems normal. Val grind’s output is saved in the log file called V A R L O G.

.

Then, we use the tail command to see the last 15 lines in the log file.

The log file says that 20 bytes are lost. In fact, the log file tells us the lost memory is allocated by line 35 of the program.

Why does the program leak 20 bytes? Because the program allocates an array of five integers. Each integer is 4 bytes. As a result, the program leaks 20 bytes.

At the very bottom of the log file, it says the program has an error.

Please understand that memory leak is not acceptable. You need to use val grind to check whether your program leaks memory or not.

Some people think memory leak can be ignored because programs may appear working for a while.

The reality is that memory leak can cause a computer to stop working. This program leaks memory really fast. Please do not do this on Purdue computers.

If you type this program and run it on your own computer, your computer will freeze and you have to reset the computer.

This example shows that memory leak is a serious problem and thus your program must not leak memory.