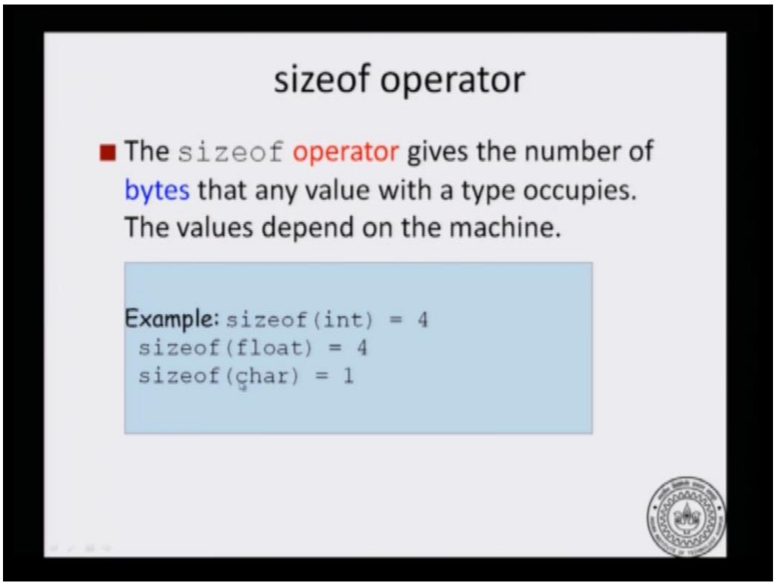
Introduction to Programming in C
Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 37

In this video, we will see the sizeof operator, which is a slightly advanced topic, in relation to pointer arithmetic. This is explaining, how pointer arithmetic freely works. And it is also important to understand one topic that, we will see later on and called malloc.

(Refer Slide Time: 00:23)



sizeof operator

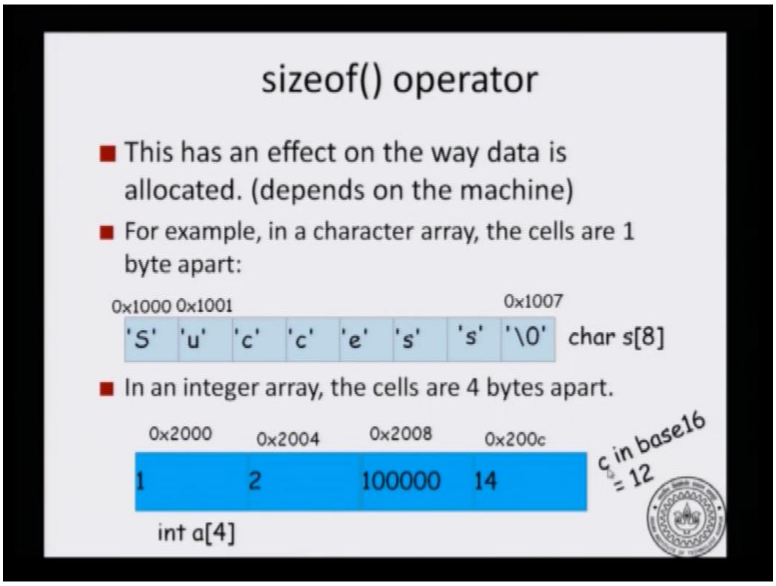
- The sizeof operator gives the number of bytes that any value with a type occupies. The values depend on the machine.

```
Example: sizeof(int) = 4
 sizeof(float) = 4
 sizeof(char) = 1
```

So, the sizeof operator and note that, it is an operator and that highlighted that in red, it looks like a function call, but it is not. The operator gives the number of bytes that any value with the given type occupies. So, sizeof is an operator which takes the name of a type of as an assigned argument, it can also take other kinds of arguments, we will see that. So, you could ask, what is the sizeof an int? What is the sizeof a float? What is the sizeof a character?

And the answer, the value that it will come out to be will depend on some particular machine. So, the reason why we use the sizeof operator is that, it helps you to write the code that is general enough for any machine, we will see, what that means? So, right now you just returns you the size of any given data type.

(Refer Slide Time: 01:22)



sizeof() operator

- This has an effect on the way data is allocated. (depends on the machine)
- For example, in a character array, the cells are 1 byte apart:

0x1000	0x1001						0x1007	
'S'	'u'	'c'	'c'	'e'	's'	's'	'\0'	char s[8]

- In an integer array, the cells are 4 bytes apart.

0x2000	0x2004	0x2008	0x200c
1	2	100000	14

int a[4]

c in base16 = 12

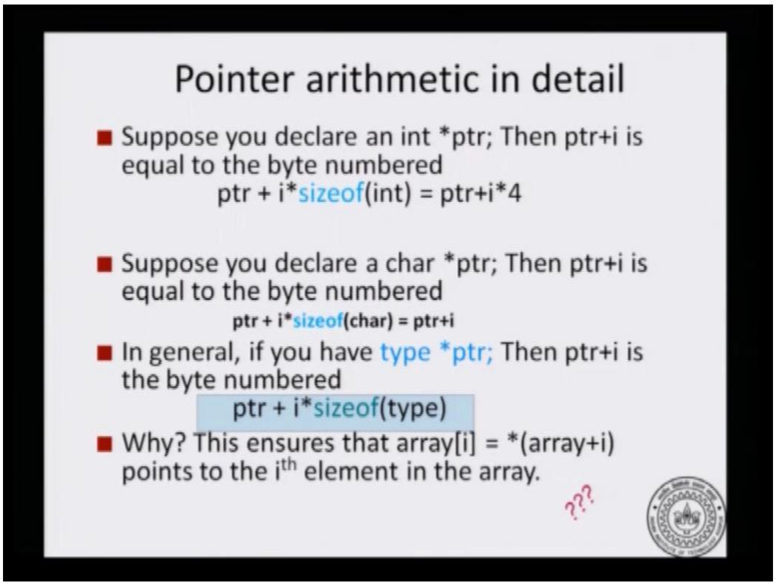
So, the sizeof operator has an effect on the way data is allocated and the way details allocated depends on the machine, we will see that. So, for an example if you have a character array, the cells are 1 byte apart. So, sizeof operator returns you the number of bytes that a data type occupies. So, in the case of a character, the character occupies 1 byte. So, if you have a character array declared as char s 8, what you have are, 8 cells and each of those cells occupy a width of 1 byte.

So, let us say that the character arrays starts at hexadecimal address 1000. So, the next cell will be at the next byte, which is byte address hexadecimal 1001. And this goes on until the last cell which is hexadecimal 1007. So, this contains a null terminator character array with the letters s u c c e s s and then followed by a null. What happens with an integer array? So, in an integer array, sizeof an int is 4 bytes. So, the successive elements of an integer array are 4 bytes apart.

So, let us say that I have declared an integer array as int a 4. This means, that a 0 let us say, it starts at hexadecimal address 2000. Then, a 1 will start 4 bytes away, because the size of an int is 4 bytes. So, it should start at address hexadecimal 2004 and I have sort of indicated it pictorially I want to say that a character is a narrower data type than an integer, an integer occupies 4 bytes. So, the next integer cell, the next integer element in the array will start at hexadecimal address 2004.

The third element will start at hexadecimal address 2008 and the last at hexadecimal address 2000 c. So, notice that I should have started it at 2012. But, 12 in hexadecimal addressing scheme is c. So, in base 16, c is the same as decimal 12. So, that is why I have written it as hexadecimal 200c.

(Refer Slide Time: 04:18)



Pointer arithmetic in detail

- Suppose you declare an int *ptr; Then ptr+i is equal to the byte numbered
  ptr + i*sizeof(int) = ptr+i*4
- Suppose you declare a char *ptr; Then ptr+i is equal to the byte numbered
  ptr + i*sizeof(char) = ptr+i
- In general, if you have type *ptr; Then ptr+i is the byte numbered
  ptr + i*sizeof(type)
- Why? This ensures that array[i] = *(array+i) points to the i^{th} element in the array.

???

Now, let us look at pointer arithmetic in greater detail with our current understanding of the sizeof operator. So, suppose you have an int star pointer. If you have an int star pointer and then you want to say that ptr plus i is equal to, what it should? So, notice that plus make sense, when you are navigating within an array. So, ptr is let us say pointing to some cell within the array and ptr plus i should go to the ith cell after ptr, that is what, it should do. Now, the ith cell after ptr means the ith integer after ptr.

So, we should skip 4 i bytes in order to reach the ith integer cell after ptr. So, thus is what we have written here ptr plus i is the byte number, ptr plus i star sizeof int, the machine addressing goes in terms of bytes. So, in order to jump to the ith integer cell, we have to know, how many bytes to skip? And the size of an integer is 4 bytes. So, this means we have to skip ahead 4 i bytes, in order to reach ptr plus i.

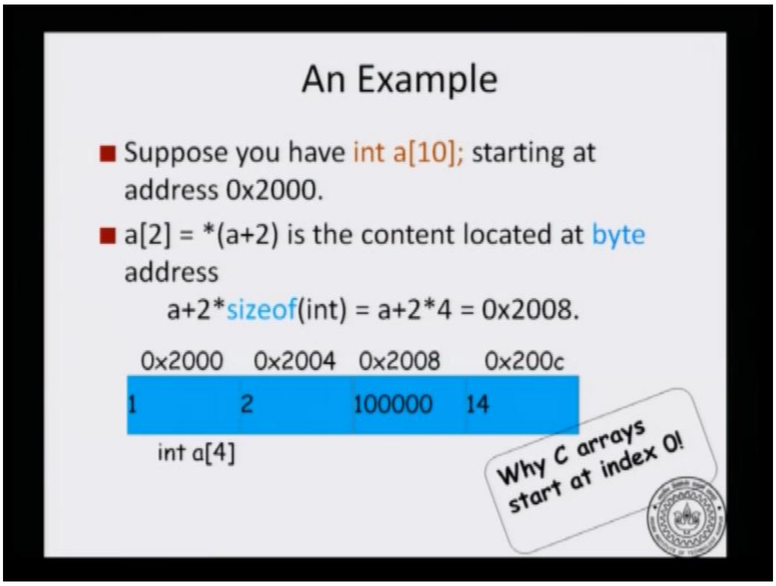
Now, if have we declared character star ptr, then ptr plus i is supposed to jump to the ith character after ptr size of a character is 1 byte. So, ptr plus i star size of character would be ptr plus i star 1, it is the same as ptr plus i. So, notice that let us say that, machine understands only byte addresses. So, in order to execute ptr plus i correctly, we have to tell which byte should I go to, should the machine go to? And in order to do that, you utilize the sizeof operator. So, since we have declared character star ptr, you know that it is sizeof character, i star sizeof character those many bytes I have to skip.

In the previous case, I have declare int star ptr. So, in that case I have to skip i star sizeof int, in order to reach the correct cell. So, here is the actual reason why ptr plus i would magically work correctly. Whether, it was an integer array or it was a character array? This is because, at the back of it all, you translate everything to byte addresses using star sizeof whatever type. So, in general if you have type star ptr, then ptr plus i is the byte number ptr plus i star sizeof type.

So, this type is the same as the declared type of pointer, ptr is a pointer to that type. Therefore, you multiply it with sizeof type and this is the general formula for pointer arithmetic. Now, one of the side effects of that or one of the consequences of this kind of addressing is that, array plus i is star of array plus i and it will correctly jump to the ith location in that array, regardless of whatever type the array was. Why is that?

Because, array plus i is then translated to array plus i star sizeof whatever type the array has been declared to be. So, you will correctly jump to the byte address corresponding to the ith element in the array. So, here is how array arithmetic in c works, in full. What do we mean by this? Let us see that with the help of an example.

(Refer Slide Time: 08:30)



An Example

- Suppose you have int a[10]; starting at address 0x2000.
- a[2] = *(a+2) is the content located at byte address

  a+2*sizeof(int) = a+2*4 = 0x2008.

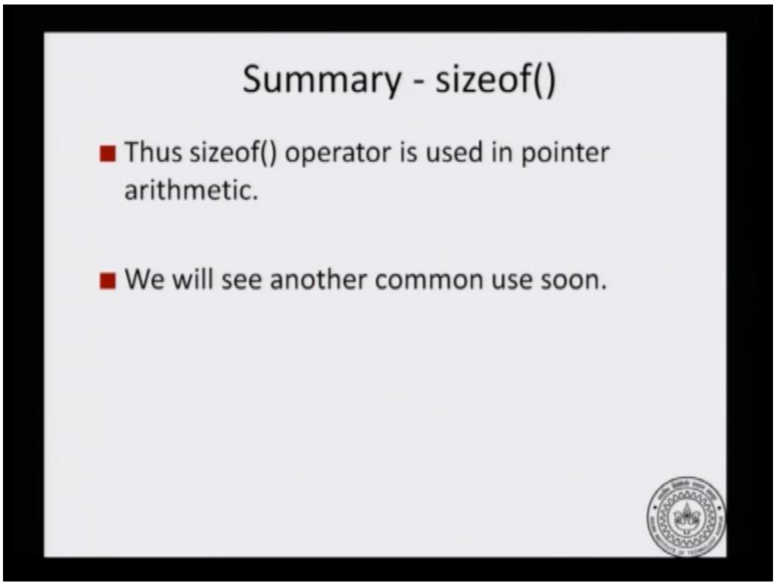
0x2000
0x2004
0x2008
0x200c
1
2
100000
14
int a[4]
Why C arrays start at index 0!

Suppose, you have an integer array declared as int a 10 and it starts at the address 2000. And I want to know, how is it that you get this third element of the array a of 2. So, a of 2 we know is star of a plus 2, a is a pointer to the first element of the array. And you have to now understand, how plus 2 is executed? So, plus 2 should be the content located at byte address a plus 2 star sizeof int. Why is this? A is been declared as integer array. And in c, integer array has the same type as int star. So, a is a pointer to int. Therefore, we know that, we have to do a plus 2 star sizeof int.

Whatever the argument is, it will do 2 star sizeof the type pointer 2 by that pointer. So, we will do a plus 2 star 4, which is hexadecimal address 2008, a was 2000. So, if you have an array int a 4, let say and it started at address 2000, then you will jump to array address 2008. And this is the reason, why c arrays start at index 0? Because, it is a very easy formula, a of 0 would be star of a plus 0, which is simply star of a.

In that case, you have the consistent explanation, that the name of the array is a pointer to the first element of the array, you do not need a special rule to do that. Think of what would have happen, if arrays started at 1. Then, a of 2 would be a plus 1 star sizeof int. So, a of n would be a plus n minus 1 times sizeof whatever and that is an uglier formula than what we have here. So, it is better for arrays to start at location 0, because it makes the pointer arithmetic easier.

(Refer Slide Time: 11:01)

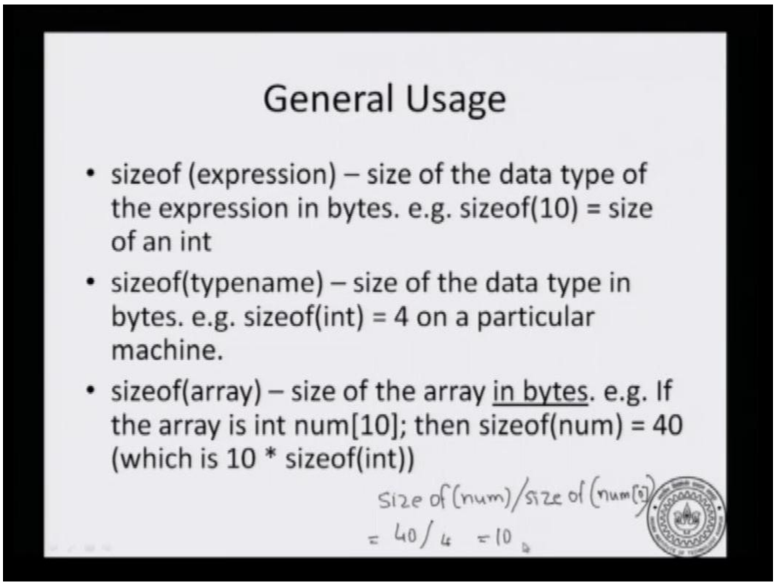


Summary - sizeof()

- Thus sizeof() operator is used in pointer arithmetic.
- We will see another common use soon.

So, in summary the sizeof operator is used in pointer arithmetic and we will see one more common use of the sizeof operator, very soon.

(Refer Slide Time: 11:13)



General Usage

- sizeof (expression) – size of the data type of the expression in bytes. e.g. sizeof(10) = size of an int
- sizeof(typename) – size of the data type in bytes. e.g. sizeof(int) = 4 on a particular machine.
- sizeof(array) – size of the array in bytes. e.g. If the array is int num[10]; then sizeof(num) = 40 (which is 10 * sizeof(int))

size of (num)/size of (num[0])
= 40/4 = 10

So, the general usage is you can give size of an expression. What will it do is, it will take the type of that expression. So, if I say sizeof 10, then 10 is an int,. So, it will execute sizeof int and let us say that, on a particular machine it is 4 bytes. Similarly, you could also say sizeof type name. So, for example, I could say sizeof int. Rather than giving an integer as an argument, I could also say sizeof int, where int is the name of the type and it will return 4 on some particular machine.

A less common usage is, you could give sizeof array, if the array is some particular array and it will return you the size of the array in bytes and this is important. It will not return you exactly the number of elements in the array, it will return the total size of the array in bytes. What do I mean by that? If I say, int num 10 and then I say sizeof num, it will return is 40 because, there are 10 integers, each integer occupying 4 bytes.

So, in order to calculate the number of elements in the array, for example, you could do the following, you could say sizeof num divided by sizeof num 0. So, this would evaluate to 40 divided by 4 which is 12. So, size of the operator on the array does not exactly give you the number of elements in the array, it will give you the total number of bytes in the array. But, if you also know how many bytes, the particular element in the array occupies, then you can easily figure out the size of the array, in terms of the number of elements.

So, also note that c does not say that, an integer is 4 bytes or float is 4 bytes and so, on. What it specifies is the relationship between the sizes of various types and we will not get in to it, right now. But, just keep in mind that the size of a particular type is depended on, which machine you are running the code on.