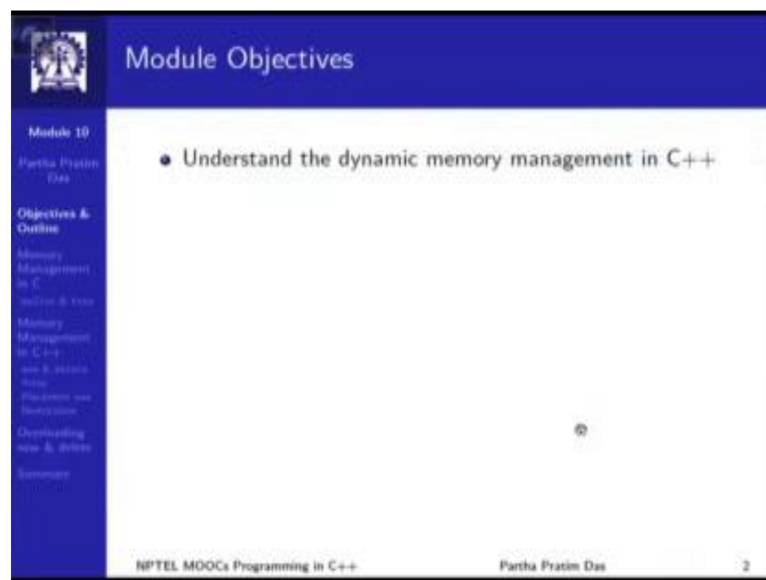
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 17
Dynamic Memory Management

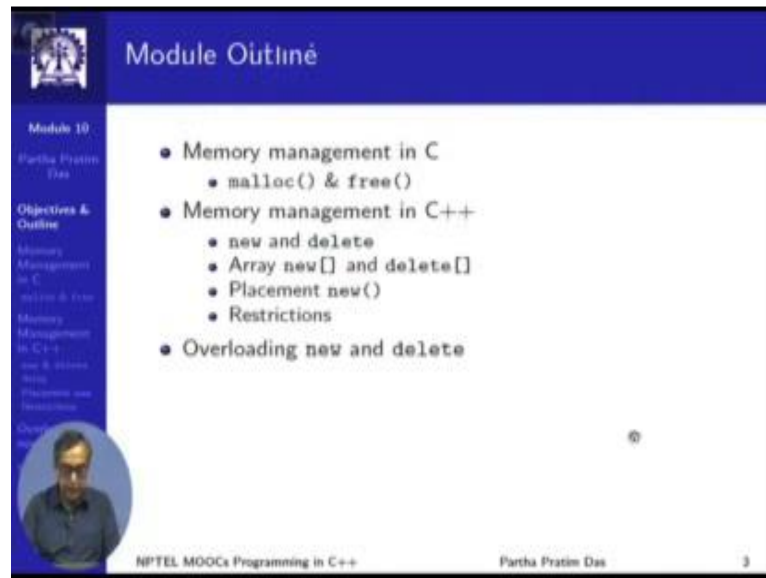
Welcome to module 10 of programming in C++. We have been discussing the precidiaral extensions of C for C++, this is the part of better C of C++, and we have already discussed several features starting from const, inline function, reference parameter and so on. And this will be the last in this series. Today, we will discuss about dynamic memory management in C++.

(Refer Slide Time: 01:05)



So, we are trying to understand the dynamic memory management capabilities of C++ over what already exist in C.

(Refer Slide Time: 01:16)



So, this will be the outline, as usual, you can see the outline on the left border as well. The memory management in C, we will recap with malloc and free, and then we will talk about various ways that the management can be done in C++, and how is that better than what we can do in C.

(Refer Slide Time: 01:40)

So, we will start with an example on the left column, you can see a very simple program which is dynamically allocating a memory. So, if we look at this the place where the allocation is happening. So, p is a pointed to an integer and we are using the malloc function which will allocate the memory on heap and the malloc function is available in stdlib dot h. So, you have included that header. And as you are aware malloc needs the size of the memory to be allocated, which is basically the number of bytes that I need for storing the data that I want to keep.

Since, I want to keep an integer, we store the size of we pass the size of int to malloc this is what we are doing here. Malloc in return, so this is this is a point we have highlighted it below also. And with this parameter, malloc will dynamically allocate the appropriate number of bytes as I consecutive junk they all will be together as a consecutive junk on the heap. So, this is an important point that here when we usually talk of dynamic memory allocation, we are talking of allocating the memory in certain segment of the memory, typically known as heap or some places this is called as a free store, this is the memory that we can use freely for your purpose.

Now, once a call to malloc is done then malloc returns a void star pointer, malloc will return a void star pointer that because malloc does not know what type of data are you going to store in the memory that malloc is allocating for you. So, malloc will return you a point at to an unknown type of data by a void star. And it is a responsibility of the programmer as in here where we will need to cast that pointer with the appropriate type, since we want an integer to be stored look at the left hand side, we will cross this as int star. This will give us, so with this, what will have is we will have something like this; this is my p and. This is the memory that has been allocated. So, this is a pointer this memory has been allocated.

So, if the address where this memory has been allocated is 200, then p will have a value 200. And then I want to dereference that is I want to traverse the point at p and store a value at this integer location. So, as I do star p a sign 5, 5 will go in to this memory location, which can subsequently be used in the print statement, and this will print the value 5. So, we all understand that and finally, when I am done when we do not need that dynamically allocated memory anymore, we will free it up by a call to the free function

this will be released by a call to the free function. And this will be released from the heap from where this was originally allocated. So, this is the mechanism of dynamic allocation of for a variable using malloc and free functions available in the standard library.

Now, certainly due to backward compatibility, C++ programs will also be able to use it. So, as we have already explained several times. Now the header name will become C stdlib, this will give everything in the std name space. So, we are saying using name space std that is we will every standard library symbol is prefixed with a std colon colon as if. And we write exactly the same code as in the left hand side. If you compare them side-by-side they are exactly the same code, and this code will run in C++ exactly at is does in C.

So, what we understand is the dynamic allocation and d allocation or management of memory, using malloc and free can be done exactly in the same way in C++ as well. Here I have been discussing about malloc for allocation alone, you know there are other allocation functions like calloc, and realloc; realloc is often used for extending a given memory and so on. So, these functions can also be equally used in C++, but we will not show explicit example of them, we will just restrict to malloc.

(Refer Slide Time: 06:42)

Program 10.02/03: operator new & delete: Dynamic memory management in C++

Module 10

- C++ introduces operators new and delete to dynamically allocate and de-allocate memory:

malloc() & free()

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    int *p = (int *)malloc(sizeof(int));
    *p = 5;

    cout << *p;

    free(p);

    return 0;
}
-----
5
```

- Function malloc() for allocation on heap
- sizeof(int) needs to be provided
- Allocated memory returned as void *
- Casting to int * needed
- Cannot be initialized
- Function free() for de-allocation from heap
- Library feature – header cstdlib needed

Operatorsnew & delete

```
#include <iostream>

using namespace std;

int main() {
    int *p = new int(5);

    cout << *p;

    delete p;

    return 0;
}
-----
5
```

- Operator new for allocation on heap
- No size specification needed, type suffices
- Allocated memory returned as int *
- No casting needed
- Can be initialized
- Operator delete for de-allocation from heap
- Core language feature – no header needed

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

Now, let us take a look at what C++ offers besides malloc and free that already comes to C++ from c. So, the program on the left hand side is the program that we have just seen, the C program using malloc and free. The program on the right hand side is using C specific mechanism of C++. Focus on this line where a memory is being allocated. C++ introduces a new operator; the name of that operator is operator new or new simply. So, that if you write new and write a data type rust after this operator new then new will do the same task that malloc was doing that is new will also request the memory manager of the system to give it some free memory from the heap and return you the pointer to that memory. So, the basic functionality of new is exactly like malloc. But there are several distinction, several differences that exist between the way new needs to be used and the way malloc is traditionally used or the malloc can be used.

First thing is malloc needs the size of the memory that we want to be passed as a parameter to malloc, new does not need anything like that. New in contrast, actually takes the type of the variable that I want to create the space for the type to be passed as a parameter. Passing a type as a parameter is a relatively difficult concept for you at this stage, because you have never seen a type being passed. So, for now, just take it by heart that new is an operator, which instead of taking a value, it can actually take a types. So, here new operator is taking the type int. And since it takes the type int, it can internally compute, how much space it requires, it can internally do the size of on this int that malloc needs to be explicitly passed, and therefore, it does not need any size specification to be given. So, in malloc, you need to give the size in new, you do not need to pass any size.

The second major distinction is that the malloc since it does not know what type of data you are going to keep in the memory that it is allocating, it cannot give you a pointer of an appropriate type. It gives you a pointer which is void star pointed to some unknown type, which is we have already discussed you know. But in new, as I have just mentioned new takes the int type as a parameter. So, new knows that you are going to keep an in type value in the memory that new is allocating therefore, new will return you a pointer of the int star type, the actual type that you need. So, that is a that is a something very very interesting because here the example is showing with new if you now do a new with double it will return you a double star pointer. If you do new with a say a structure type

complex, then it will return you a complex star kind of pointer. So, it new will knows what type of value you are going to keep and therefore, it returns to a pointer of that appropriate type. So, note on this point that, since malloc returns you a void star, you will need to cast it to int star; new in contrast will directly give you an int star pointer and no casting is required.

Next after the allocation was done, in this case, we went ahead and put an initial value in the location that has been allocated, we had a sign 5 to star p. Now using malloc, there is no way to allocate and initialize together. New gives you even that facility that is I can if I want I can pass this initial value five as I am doing here, so that when new returns me the pointer, it will not only give me the pointer of the location allocated, but that location already carry the initial value five that I want there. So, if we just see the dynamics, if I just draw them again this is p on this side of the program and this is the allocation. So, once this is done, you will get a state like this where you do not know what value exist. And then you will do star p assigned five which will take five and put it here. So, there is no initialization in this case, you are making an assignment afterwards through the second statement.

Whereas, here when this will get executed, when you get back the pointer p you get back the whole thing in this situation that the pointer p is pointing to their allocated location. And the value already is 5 so that is a very good advantage that in malloc initialization is not possible in new the initialization can be done. Of course, you have the choice of if you want that you would like to just allocate and not initialize, you can even do that all that you will need to do is write it as new int and not give the initial value; in that case it will give you an uninitialized at malloc gives you.

Finally, when you are done, you will if you had allocated the memory by malloc, you need to free it release it by a call to the free function. Here, you have another operator which is also new for C++, another operator the name of this operator is delete. So, you write the operator delete and then put the pointer. So, this will release the memory back to the system as free does. They are very similar in terms of. So, free was deallocating in the memory from the heap, and operator delete will similarly deallocate the memory from heap.

In all this discussion of a new and delete, please remember that I am consistently saying that malloc and free are functions, whereas, new and delete are operators. So, while we discussed about operator overloading, we gave you several distinctions between an operator and a function. So, those distinctions will exist between operator new and function malloc, or operator delete and function free and we will make use of some of those distinctions as we go forward. Finally you can also note that malloc and free are not part of the core language. So, they come from the standard library C stdlib, whereas, new and delete are part of the core C++ language therefore, to do new or delete you do not need any special header to be included.

(Refer Slide Time: 14:53)

Now there is something very, very interesting. As we have the operator new and delete, similarly we can also write the operator functions for new and delete. We know that every operator has a corresponding operator function. So, it is possible that you use the operator function directly also. So, in this example, again on the left hand side, we have the same malloc example with the free. On the right hand side, instead of operator new, what we are showing is the use of the operator function new, the corresponding function the difference, if you want to me to recall from the previous slide then when you use operator new you just write new and then pass the type. Whereas, when you use the operator function corresponding to new that is operator new function, it is pretty much

like malloc. So, you need to pass the size you are no more passing the type, you need to pass the size, you need to cast the pointer back.

So, in C++, it is possible that you can also write malloc kind of call using the operator new function. And correspondingly you could also write in place of operator delete, which we had invoked earlier this is how you invoke operator delete, instead you could also invoke it in the style of free function. You can say operator delete which is a corresponding operator function and pass p as a parameter to the operator delete function. So, as malloc allocates on heap, operator new function will also allocate on heap as free deallocate; from heap operator delete function will also deallocate from heap. But please keep in mind that operator new and the operator new function are two very distinct entities, and there are major differences between operator new and the function operator new.

So, at the example up to which we have discussed no difference can be seen it appears as if I can either use operator new or I can use the function for operator new. If they are equivalent, they are just giving us the same effect this is true as long as we use the built in types. But, the moment we use some of the user define types we will see that the v operator new, the operator version works, and the way the corresponding function version works become very different. We will discuss them when we come to that stage, but we will explode that more, but right now we just I just want you to note that there is a major difference between them.

(Refer Slide Time: 18:12)

Now, let us move on and look at what we dynamically allocate very often in C, we often allocate arrays. So, if we want to allocate arrays, we use the same malloc function, you all are familiar. So, this is the same malloc function. The only difference is we saying its size of int is multiplied by 3 which any experienced C programmer will understand that what we are trying to say is I want three consecutive areas of size int which basically mean that here this pointer should actually eventually give me an array of 3 int elements.

This is how you read it, but if you look into what value malloc gets, malloc necessarily does not get to know what is size of int, or what is the value three malloc gets their product. So, if size of int is 12, malloc will I am sorry, if size of int is 4, the malloc will get a value 12. The malloc has no way to know that weather these are for 3 integer or it is for twel12e unsigned characters and so on. It just gets a total cumulative value.

In contrast, operator new and this has a different name we call it array operator new. This array operator new will like before take the type which is a type of the element which is int, but then within a square bracket, it takes the number of elements that are required. So, it will know how many elements of int type to be created in the memory that needs to be allocated. So, behavior wise, it is very similar to malloc, but specification wise, it is making it very clear that you can even see the uniformity of the int that you actually

wanted to do this int array of three integers. And you literally write the same thing and as before it will return you not a void star pointer which needs to be cast in case of malloc, but it will return you an int star pointer.

Similarly, when we have done, you do free in case of a memory allocated by malloc, but in case of a memory allocated by array new, where you have used this number of elements, you are pass the number of elements after that type, you will need to delete it by array delete. The difference with the previous form is specifically need to put the array operator after the delete key word and before the pointer. And then the delete array delete operator will release the memory and to the heap. And please note that operator new and operator array new are different operators. Similarly, operator delete and the operator array delete are different operators; they can be handled extremely differently and we will show some examples of that.

So, this is the second type. So, it is advised that if you are dealing with individual data items then you just to use operator new and operator delete, but if you are using arrays of data items then you use operator array new and operator array delete. So, that system explicitly knows when we are dealing with an array, and when you are dealing with the single data item. This distinction was not possible in C using the standard library malloc free function, but here it is semantically gets absolutely clear.

(Refer Slide Time: 22:14)

Now, we talk about yet another form of memory allocation in C++. This is known as placement new. This is another form of new operator, which allocates memory, but it has something which is very different in fundamentally in the concept. When we deal with malloc or for that matter operator new or operator array new, we want to allocate the memory and that memory we want has to come from heap that is that memory has to be negotiated with the memory manager of the system and should come from the free store area free store segment of the memory. So, in that way, if we allocate any variable or an array in that manner, then certainly I do not have a control in terms of in which address or in which location this variable gets created.

But I may want to and I will slowly explain the reason why I want to do this. I may want to do the same allocation, but I may want to do them in a memory address which is known to be which I pass to this particular allocator. The situation is let us suppose that i have a buffer a buffer is nothing but you can you can just read the specification, you can read the definition of the buffer. It says the name is buff it is of type unsigned character and the size is or the number of elements of this buffer are size of int into 2, which means that if I am on a 32 bit machine then very typically size of int is 4. So, it is basically eight characters eight bites consecutive that I have as a buffer.

Now, what you want I want to dynamically create an integer, but I do not want to create that integer in the free store. I want to create the integer in the buffer itself. So, if i try to let us say if I try to drop the buffer then say this is the buffer. So, these are the 8 bytes of the buffer this is buff, and since integer is of 4 bytes, so it is possible that I can take these 4 bytes and these 4 bytes to keep one integer here and another integer here and so on. That is these four consecutive bytes starting from let say address if this address is 2 0 200 then this address is 203, similarly this address will be 204, so this address goes up to 207, from 200 to 203 in that buffer I want to keep an integer; and the other one, I would not to keep.

So, I want to still dynamically allocate, I want to dynamically allocate look at this line, I want to dynamically allocate an integer and get that pointer as p int. But I have between the new and the int type, I have put the name of this buffer which tells me that please do not allocate it in a free store allocate it from this buffer that is buff is basically a pointer So, value of buff is twent 200. So, what will this 2 p int, p int will we allocate it the address from this buffer address of 200. And it will think of this part of the buffered as its integer. Similarly, if I next do q int as in here, I am again allocating an integer, but I do not do it as at buff, I do at buff plus size of int. So, buff was 200 size of int is 4. So, buff plus size of int is 204. So, if I do that allocation then q int gets a value 204 that is it points here, and therefore, to q int it is as if this is the other integer.

So, it is doing the allocations exactly in the same way, but with the difference that the allocations are not happening from the free store, it is happening from a buffer that already exist. Now, this buffer itself here I have taken the buffer to be of an automatic type, which is local to the function body, this buffer itself could be dynamically allocated in the free store or otherwise that we are not concerned with that. But that the reason we call this is a placement new, because it is not only doing an allocation, but it is a doing an allocation and placing it where I want it to be. So, this placement is the fact that I am not only doing the allocation, but I am passing a pointer saying that this is where memory is available you go and allocate it there, you go and place the object there.

Now, you will certainly argue that this is this could have been possible in C++ by writing various kind of pointer manipulations, why did I need to really do this, you will

understand that when we talk about the dynamic allocation of user define types. For now, you just have to note that there are possibilities of doing a placement new. The only difference in case placement new from the other two forms of new is that for placement new you have provide already provided that buffered. The memory was not dynamically allocated. Since, it was not dynamically allocated from the free store there is no sense of doing a delete there, because the memory manager did not actually get you that memory.

So, for this kind of a new, the placement new here or for the placement new here, you do not have a corresponding delete; otherwise, you can go through the rest of the example starting from initialization right here or the use in terms of other pointers. If you go through this code this is what a little bit of point at tweaking, and it will be good to get a cestrum to that you will understand that rest of it is exactly like any other pointed manipulation. But the only difference is that the addresses are not coming from the dynamic store not coming from the free store addresses are coming from the buffer that I have provide it.