Programming in C++

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

Lecture - 18
Dynamic Memory Management (Contd.)

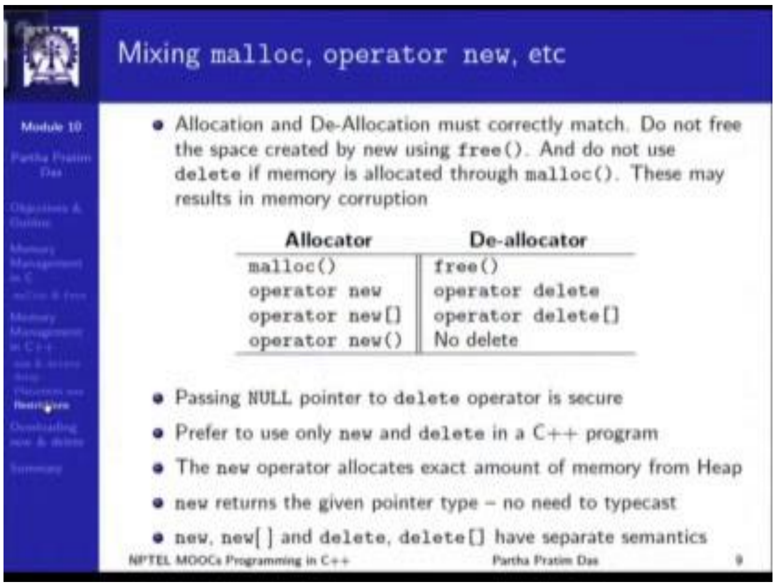
Welcome to part-2 of module 10. We are discussing Dynamic Memory Management in C++. We have done a quick review of Memory Management Dynamic Memory Management in C, using Malloc and Free.

And then we have introduced 3 pairs of operators for Dynamic Memory Management in C++, the operator new and operator delete. The operator new can take a type, and can also take an initial value and dynamically allocate a memory in free store initialize it with the value given and then return a pointer of the type of value that has been created.

If you want an array of such values to be dynamically allocated in free store we have an operator new, which is of the array kind we say, array operator new and it has a matching array delete operator. And we have finally also shown that it is possible that we can do an operator new with a given address of memory that it is not necessary that the memory comes from the allocation by a memory manger, I can have a buffer that I already have either the buffer could be in the automatic area the stack area, the buffer could be in the global area, but I can pass the address of that buffer and create objects within that buffer.

Now, we will go ahead and discuss some of the restrictions that must be followed while you are doing this.

(Refer Slide Time: 01:52)



Mixing malloc, operator new, etc

Module 10

Partha Pratim Das

Objectives & Outline

Memory Management in C

malloc & free

Memory Management in C++

new & delete

Array

Placement new

Restrictions

Overloading new & delete

Summary

- Allocation and De-Allocation must correctly match. Do not free the space created by new using `free()`. And do not use `delete` if memory is allocated through `malloc()`. These may results in memory corruption

Allocator	De-allocator
`malloc()`	`free()`
`operator new`	`operator delete`
`operator new[]`	`operator delete[]`
`operator new()`	No delete

- Passing `NULL` pointer to `delete` operator is secure
- Prefer to use only `new` and `delete` in a C++ program
- The `new` operator allocates exact amount of memory from Heap
- `new` returns the given pointer type – no need to typecast
- `new`, `new[ ]` and `delete`, `delete[]` have separate semantics

NPTEL MOOCs Programming in C++ Partha Pratim Das 9

So, since we have multiple different operators now to; and multiple different operator as well as different functions to Allocate and De-allocate memory dynamically; we will have to be careful to note that the Allocation and De-allocation of any memory will always have to be done by the Allocated, De-allocated pair.

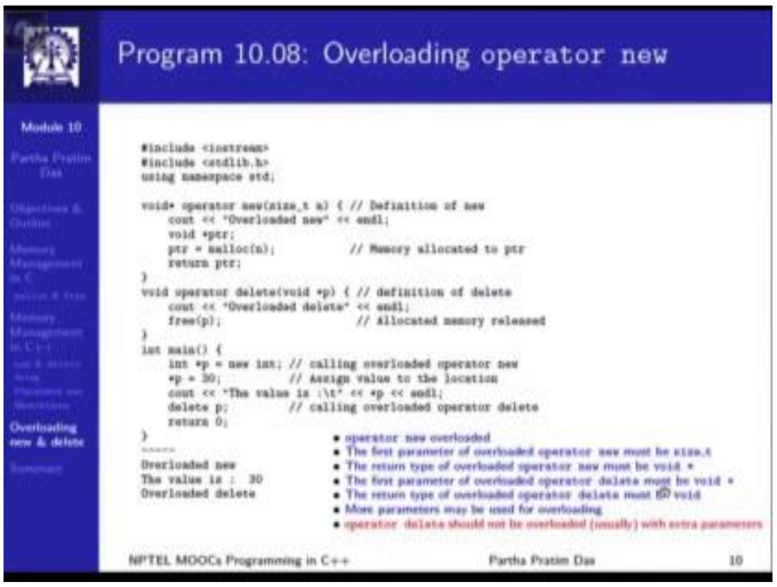
So, in the table I have listed these pairs and you will have to be very careful to follow that, that if a memory location is dynamically allocated by malloc it must be released by free. If it has been allocated by operator new it must be released by operator delete. If it has been allocated by operator array new it must be released by the operator array delete [ ] and if it has been done true placement new then there is no need for doing a delete.

Now, any kind of other mixtures will lead to ambiguity and uncertain results. For example, if you allocate a memory by malloc and try to delete it by operator delete or allocate a memory by operator new and release it by free or you allocate a memory by operator array new and try to release it by simple delete and so on, any kinds of mixture combinations of Allocation and De-allocation that you may do will lead to unpredictable disastrous results. So, be very very careful to always use them in pairs.

And while you are Programming in C++, my advice is that you always try to only use new and delete operators. It is better not to use malloc and free unless in very exceptional cases where their use is absolutely necessary. But, in a normal program you will not need malloc and free to be used you can always do it operator new or operator array new and the corresponding delete function.

So, always delete operators always try to do that and it is also possible that the delete operator can be passed in NULL pointer. So, you do not need to check if you are trying to release a memory then that memory actually exists. If that memory has been NULL, the pointer has been Null even then the delete will work safely. So, while you do this Memory Allocation Management course please keep these points in mind.

(Refer Slide Time: 04:40)



Program 10.08: Overloading operator new

Module 10

Partha Pratim Das

Objectives & Outline

Memory Management in C

malloc & free

Memory Management in C++

new & delete

Array

Placement new

Restrictions

Overloading new & delete

Summary

```
#include <iostream>
#include <stdlib.h>
using namespace std;

void* operator new(size_t n) { // Definition of new
    cout << "Overloaded new" << endl;
    void *ptr;
    ptr = malloc(n);          // Memory allocated to ptr
    return ptr;
}
void operator delete(void *p) { // definition of delete
    cout << "Overloaded delete" << endl;
    free(p);                   // Allocated memory released
}
int main() {
    int *p = new int; // calling overloaded operator new
    *p = 30;          // Assign value to the location
    cout << "The value is :\t" << *p << endl;
    delete p;         // calling overloaded operator delete
    return 0;
}
```

```
-----
Overloaded new
The value is :  30
Overloaded delete
```

- operator new overloaded
- The first parameter of overloaded operator new must be size_t
- The return type of overloaded operator new must be void *
- The first parameter of overloaded operator delete must be void *
- The return type of overloaded operator delete must be void
- More parameters may be used for overloading
- operator delete should not be overloaded (usually) with extra parameters

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

Next we look at something which is completely new in terms of C++, any corresponding concept of that in C did not exist. We have seen that operators can be overloaded and now we know that the Allocation operator new and the De-allocation operator delete are actually operators themselves, having corresponding operator functions and therefore, they can be also be overloaded. So, in this here in first I show an example of the operator new, simple operator new being overloaded. So, if you look into the definition of operator new, you will see that the parameter it takes is a size t parameter.

Now, this is what it actually expects is and just think about the corresponding call of this. So, this call will actually invoke this function. So, if you look at the call, the call does not exactly have a parameter - call has a type new int. So, if the call is saying apply operator new on type int so, but the actual operator function expects a size t, size t is nothing but unsigned int. So, int C standard library size t is type def it is a type aliased with unsigned int. So, whenever its size underscore t is written you will know that an unsigned int is meant and particularly it is size of something, most often it is a number of bytes of some structure and so on. So, size t necessarily is expecting that n the parameter here for the operator function is necessarily the size of the int.

Now, you did not specify that size off. So, the trick is done by the compiler. The compiler knowing that you want int will actually convert; we will compute the size of that int and pass it as a parameter n. So, whenever you overload operator new this size t n must be the first parameter that must be specified without that the operator new will not work. And then what we have done we have provided a body for this operator new here I have put a message just to make sure that this particular version of the overloaded operator is getting used, then I take a local pointer variable to void, then internally I am actually doing a malloc.

So, it feels as if it is operator new, but internally I am using a malloc to just allocate a raw amount of memory of the size of an int which is n now and then we have just return that pointer because it as to return the pointer. And I have already explained that operator new will actually return you a pointer of the required type that is it will return you an int star.

In this case, it does not need a casting to be done this is again a trick that the compiler will do. So, at the call time the compiler will from the type compute the size and pass it appropriately at the return time the compiler will appropriately change the type of the pointer and return a pointer of the required int type, but simply by writing this operating new function and writing a definition of that you are able to overload that.

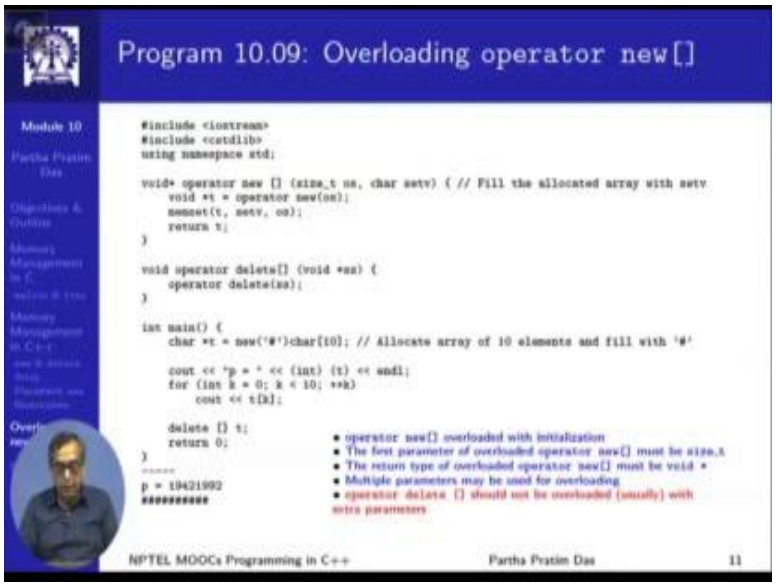
So, you can see that operator new can be overloaded the first parameter must be size t in which the compiler will put the size of the type. Return type will have to be void star this you cannot change then it will not behave as an operator new, but you can actually add more parameters to this. Here, we have not added parameter, here we have just shown a different behavior by doing an allocation ourselves and by providing a message when this particular overloaded operator function works, but you can also add more parameters and pass those parameters from the new site.

Similarly, the operator delete can be overloaded as we have done it here again the first parameter has to be void star which is the pointer that needs to be released and the return of type has to be void because; obviously, delete cannot return any computation, but we can have a different implementation here from the default operator delete that exist. So, if you run this program you will find that at the time of allocation it prints overloaded new that is, this particular cout.

Then it prints the value from this point which is 30 and finally, it prints the outputs the overloaded delete message which comes from the overloaded delete operator so, it clearly shows you that both new and delete have been overloaded and your behavior can be put.

One last point to remember here that when you overload delete usually try not to overload it with extra parameters I cannot say that this is wrong according to C++, but there is a certain context in which only you can use a delete operator which is overloaded with more parameters than the void star first parameter that it must have, and till we have come to explain those context you will not be even if you overload a delete with extra parameters you will not be able to use them everything will be shown up as a compilation error.

(Refer Slide Time: 10:49)



Program 10.09: Overloading operator new[]

Module 10

Partha Pratim Das

Objectives & Outline

Memory Management in C

malloc & free

Memory Management in C++

new & delete

Array

Placement new

Restrictions

Overloading new

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

void* operator new [] (size_t os, char setv) { // Fill the allocated array with setv
    void *t = operator new(os);
    memset(t, setv, os);
    return t;
}

void operator delete[] (void *ss) {
    operator delete(ss);
}

int main() {
    char *t = new('#')char[10]; // Allocate array of 10 elements and fill with '#'

    cout << "p = " << (int) (t) << endl;
    for (int k = 0; k < 10; ++k)
        cout << t[k];

    delete [] t;
    return 0;
}
-----
p = 19421992
##########
```

- operator new[] overloaded with initialization
- The first parameter of overloaded operator new[] must be size_t
- The return type of overloaded operator new[] must be void *
- Multiple parameters may be used for overloading
- operator delete [] should not be overloaded (usually) with extra parameters

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

So, here we show another example with the operator new. So, this is also overloading. Now, I am trying to overload operator new. So, this operator is to be used for allocation of an array of characters of 10 elements. So, naturally the first parameters still continues to be size t which is the total number of bytes that need to be allocated which can be computed here from this type and as well as the number of elements. So, size of char times 10 this will get passed as the size t os parameter and this will be done by the compiler.

The interesting point to note is now you can see a second parameter the operator array new does not have a second parameter, but we have now overloaded it with a second parameter and just see the syntax in which the second parameter has to be used. So, this is the formal parameter here and the actual parameter is specified within a pair of parentheses right after the new key word and before the type that we want to dynamically allocate.

So, this means that the character symbol hash will be passed as a value of setv and then if you look into this quote you will find that we are doing a MEM set. Let us look at the first line here we are using the operator new function we could have used malloc also, but we are using operator new function to allocate the required amount of space which is waste in this case, and we get that as a void star pointer and then using that pointer and using this setv parameter we actually fill up that buffer with the setv value.

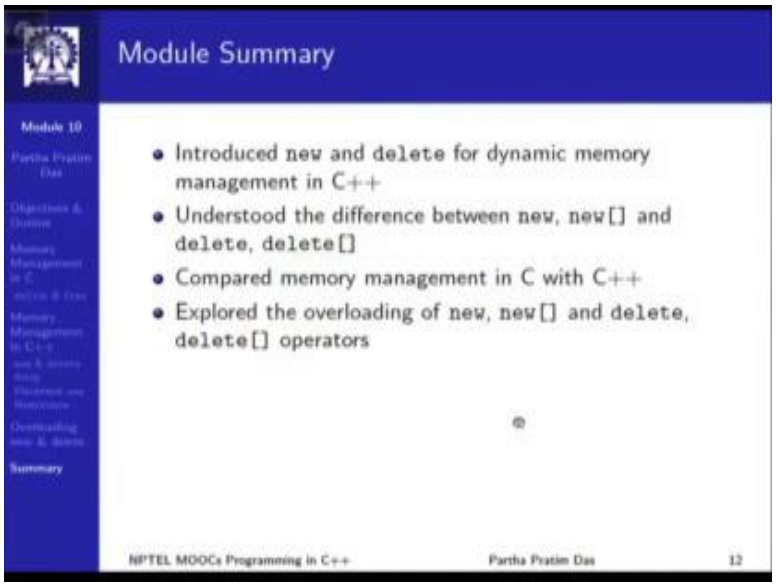
So, we start from t take a character and go waste number of times because we know waste number of bytes exist. These are basic MEM set function which exist in C, if you are not familiar with this function I would advise that you take a look at it separately in your C Programming you can try it out, but this basically takes a pointer.

So, this is like, t is a pointer to this buffer which is got dynamically allocated. It has waste number of locations and this will go and put the setv in this case the hash character in each one of them. So, the difference with the other operator array new and are overloaded array new will now be able to not only allocate a character array, but it will be also be able to fill it up with the given character at the initial time. So, it does an initialization of a different way.

So, in this way you can think of various other overloading that you may need and accordingly you may have more and more parameters in behavior and those parameters, those extra parameters you will have to be passed as a comma separated list through the point that you show here.

Similarly, here the array delete is also overloaded here though you have put any new functionality in that we have simply called the delete operator function in this overloaded array delete. So, we have shown how to overload operator new in this case for initialization and again like a single item allocated operator new, operator array new also must have a first parameter which is size t. It will always have a return type, which is void star and multiple parameters can be used for overloading an operator delete, again like operator delete if overloaded should not usually have extra parameters till we understand how those extra parameters are to be used.

(Refer Slide Time: 15:22)



Module Summary

- Introduced `new` and `delete` for dynamic memory management in C++
- Understood the difference between `new`, `new[]` and `delete`, `delete[]`
- Compared memory management in C with C++
- Explored the overloading of `new`, `new[]` and `delete`, `delete[]` operators

NPTEL MOOCs Programming in C++ Partha Pratim Das 12

In summary we have introduced the new and delete operators for dynamic memory management in C++ these are in strong contrast from C where the malloc, free and other functions are actually standard library features, not the language features here they are language features.

And we have carefully understood the differences between, different new operators and their corresponding delete operators and one basic principle that we have learnt is whatever allocation we do the allocating function or operator that we use we must use the corresponding De-Allocating function operator - malloc with free, operator new with operator delete, operator new with operator array delete and if I have done placement new then you should have no delete, and otherwise we are likely to get a whole lot of unpredictable problems.

We have also, at the end seen how to overload these new operators and delete operators, which is the advantage of having them as operators now. And we have shown how different functionality can be put in as desired by the user by overloading these operators.