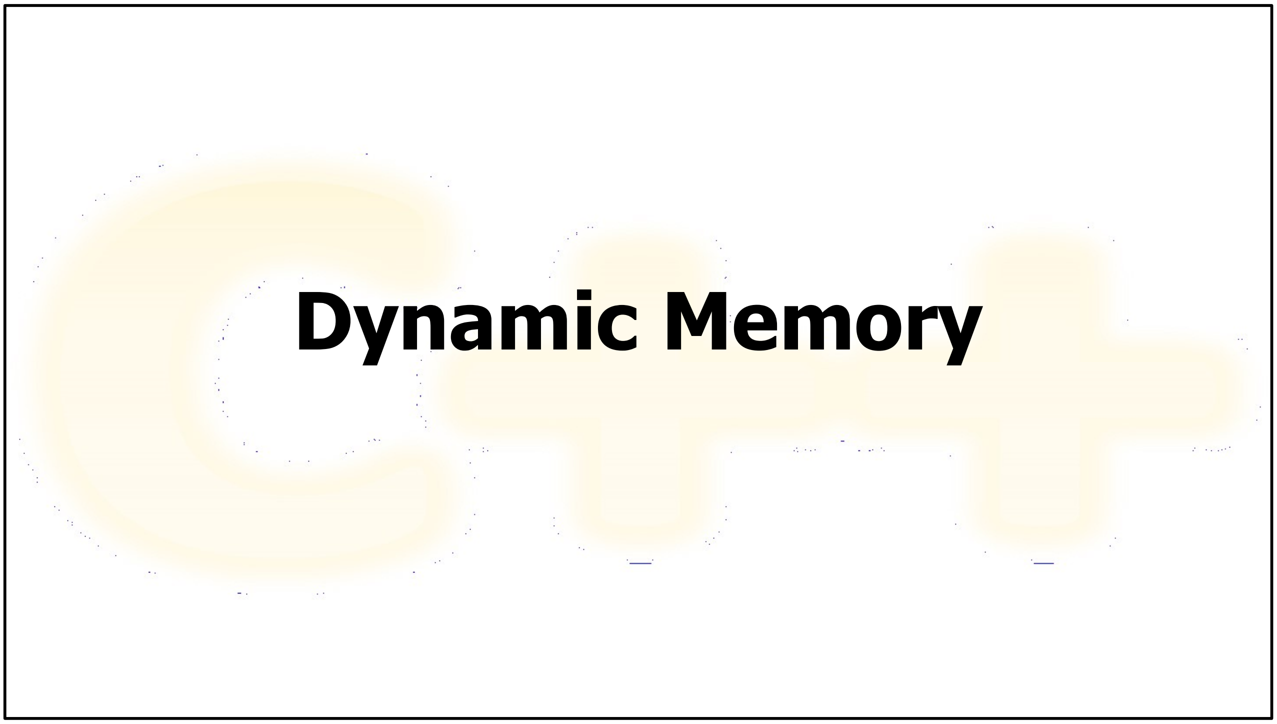
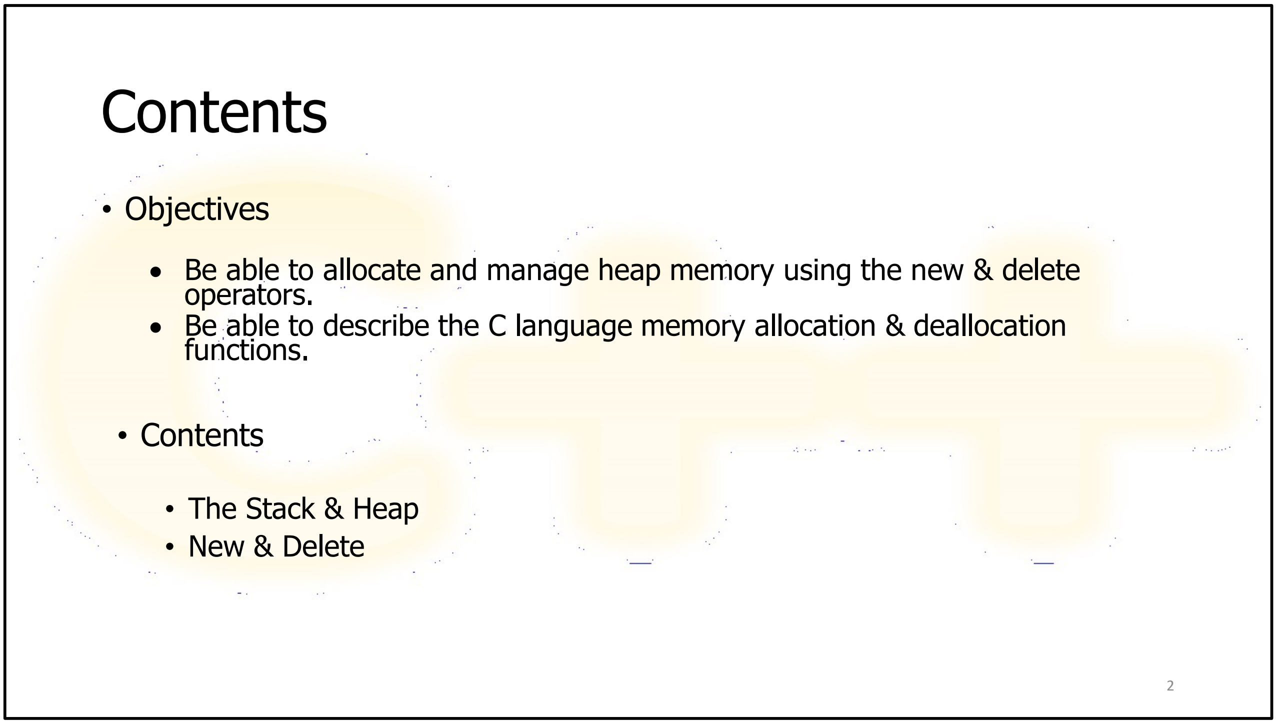
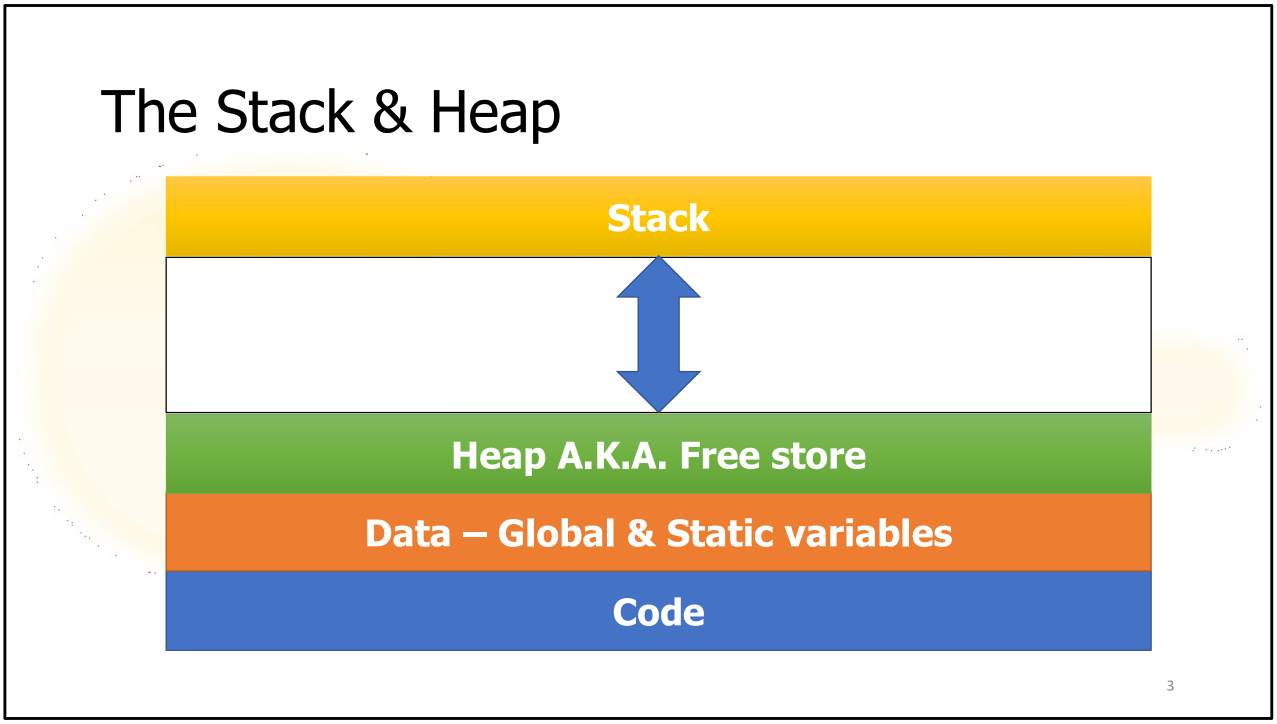


Dynamic Memory



Contents

- Objectives
  - Be able to allocate and manage heap memory using the new & delete operators.
  - Be able to describe the C language memory allocation & deallocation functions.
- Contents
  - The Stack & Heap
  - New & Delete



The Stack & Heap

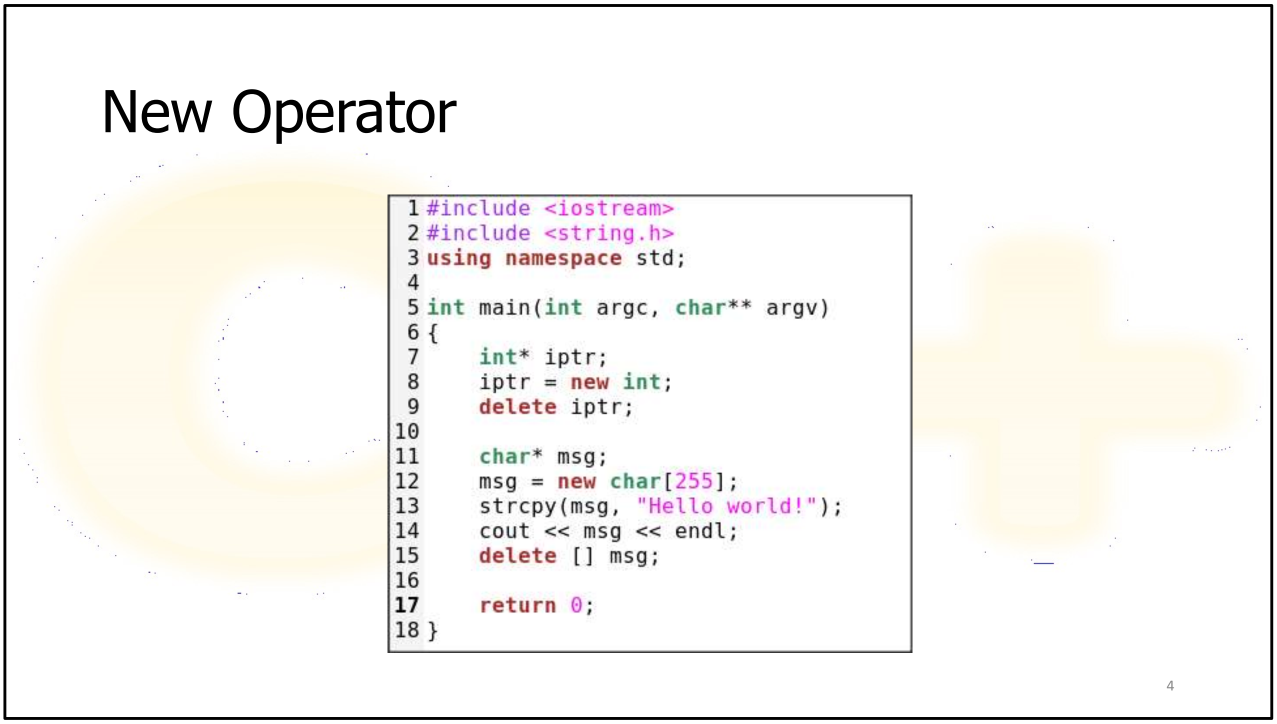
Stack

Heap A.K.A. Free store

Data – Global & Static variables

Code

C++ has 4 distinct regions for memory distribution:
Stack : This region is used for function calls' return addresses , arguments and local variables. It typically has a fixed size (1Mb on Microsoft systems)
Heap : This region is for dynamic allocation of memory (dynamic variables created at run time use this memory)
Global Variables : This is used for global & static variables defined by the programmer
Program Code : This region is for the program code.



New Operator

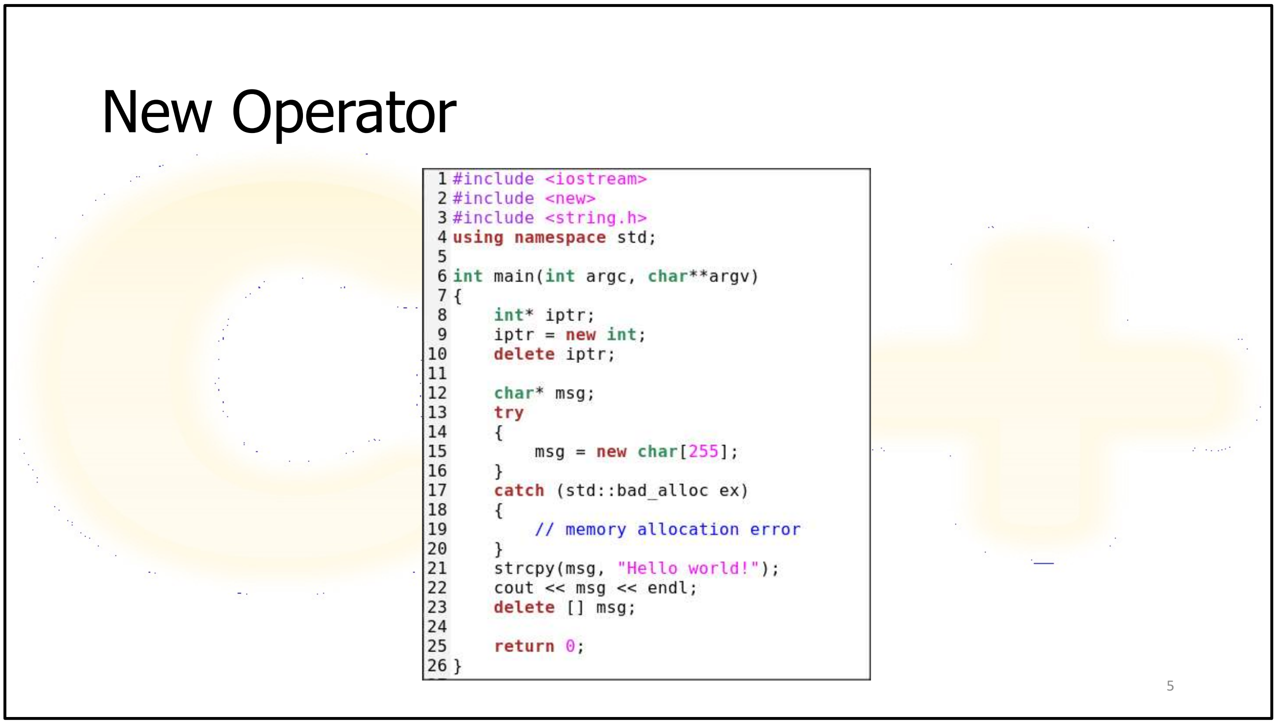
```cpp
#include <iostream>
#include <string.h>
using namespace std;

int main(int argc, char** argv)
{
    int* iptr;
    iptr = new int;
    delete iptr;

    char* msg;
    msg = new char[255];
    strcpy(msg, "Hello world!");
    cout << msg << endl;
    delete [] msg;

    return 0;
}
```

The new operator is used to allocate memory from the free store or heap. When new is called a pointer to the new memory is returned. If there is insufficient free memory to satisfy the request a null pointer is returned. New [] is used to allocate a number of objects (an array??? See later!) Memory allocated with new [] MUST be deallocated with delete [] otherwise memory leakage & possibly heap corruption may take place.



New Operator

```cpp
#include <iostream>
#include <new>
#include <string.h>
using namespace std;

int main(int argc, char**argv)
{
    int* iptr;
    iptr = new int;
    delete iptr;

    char* msg;
    try
    {
        msg = new char[255];
    }
    catch (std::bad_alloc ex)
    {
        // memory allocation error
    }
    strcpy(msg, "Hello world!");
    cout << msg << endl;
    delete [] msg;

    return 0;
}
```

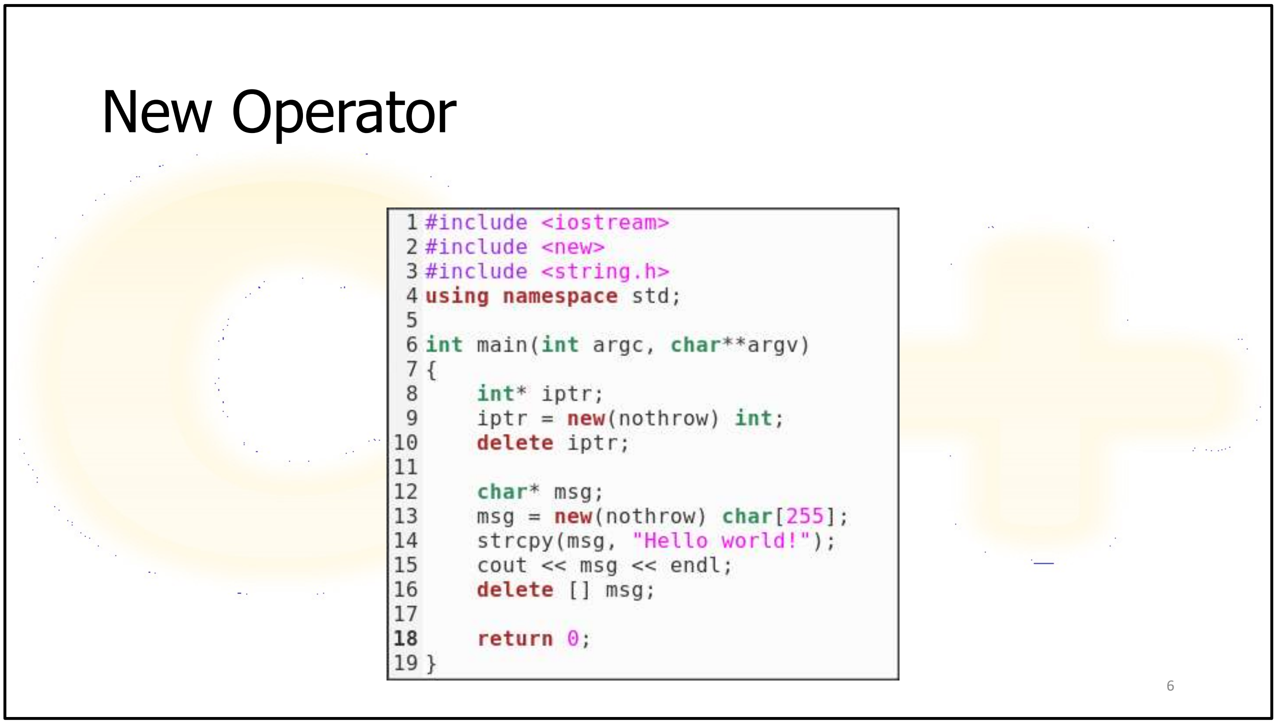
5

This simple example on the previous slide uses the standard global operator new. This has been superseded by a newer version of the operator which throws an exception if memory cannot be allocated.

The newer versions of the operators are used by including the <new> header file.

The Standard now states that operator new throws an exception of type std::bad_alloc when it fails, rather than returning a NULL pointer.

Although compiler vendors have been sluggish in adopting this change, most C++ compilers now conform to the standard in this respect, and throw an exception of type std::bad_alloc when new fails.



New Operator

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

int main(int argc, char**argv)
{
    int* iptr;
    iptr = new(nothrow) int;
    delete iptr;

    char* msg;
    msg = new(nothrow) char[255];
    strcpy(msg, "Hello world!");
    cout << msg << endl;
    delete [] msg;

    return 0;
}
```

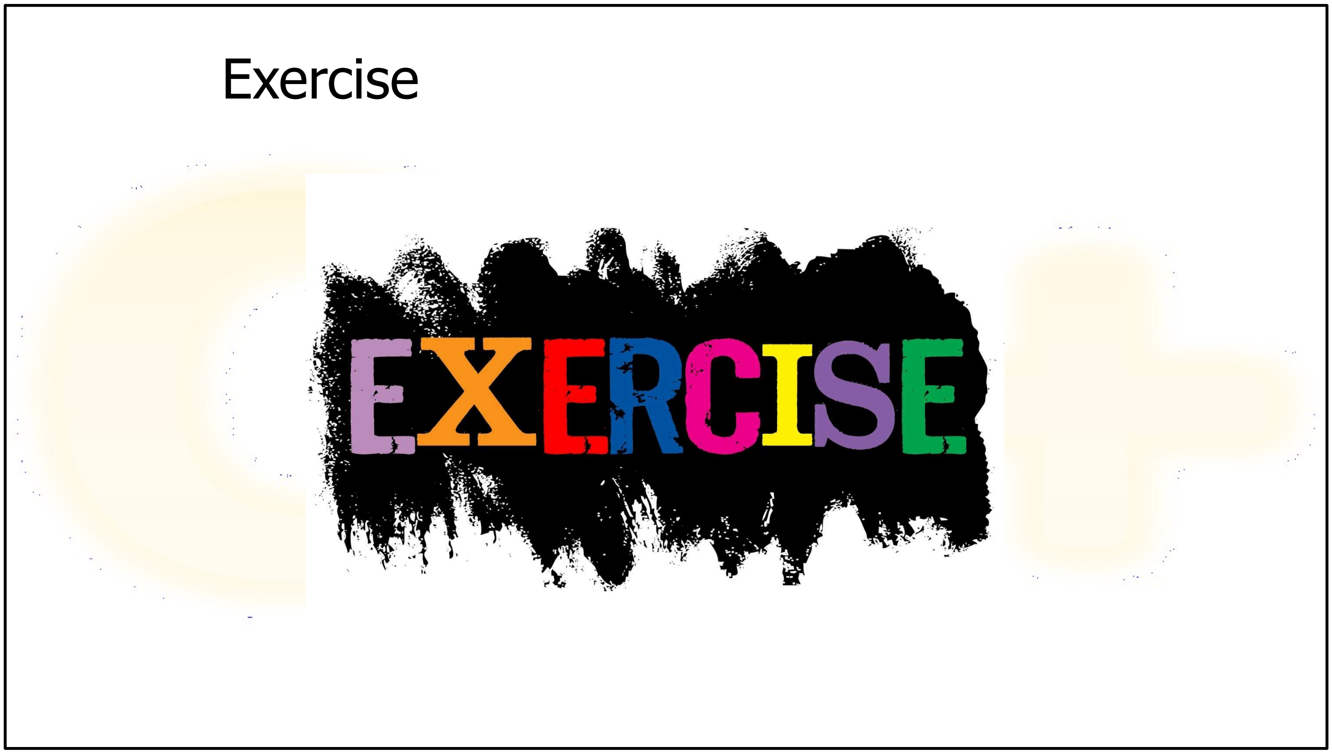
6

Under some circumstances, throwing an exception is undesirable. For example, exception handling might have been turned off to enhance performance; on some platforms, it might not be supported at all.

The Standardization committee was aware of this and added an exception-free version of new to the Standard. The exception-free version of new returns a NULL pointer in the event of a failure, rather than throwing a std::bad_alloc exception. This version of new takes an additional argument of type const std::nothrow_t& (defined in the header <new>). It comes in two flavours, one for plain new and another for new[].

Delete Operator

The delete and delete [] operators immediately de-allocate memory. The memory is returned to the freestore for re-use in future memory allocation requests. It is NOT garbage collect a-la-Java & C#. No memory compression or defragmenting is performed.



Exercise

Write a program that stores the names & ages of the delegates on this course. This information should be stored in a structure that has a string and an int member. E.g.

```
struct delegate
{
   string name;
   int age;
};
```

The structures should be stored in an array of pointers e.g.

```
delegate * students[12];
```

Each delegate will require a new of the structure. When the array is populated print out a list of all the delegates. Ensure that memory management is correct. i.e. memory is allocated & deleted correctly.