FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Lecture 5

Smart pointers

Arrays

Array: Data structure that holds a collection of unnamed objects that can be accessed by an index.

Example: int arr[5];

dynamically allocated: An object that is allocated on the program's free store.

Objects allocated on the free store exist until they are explicitly deleted.

free store (heap): Memory pool available to a program to hold dynamically allocated objects.

Security problem: buffer overflow

Operators

[] operator: The subscript operator takes two operands: a pointer to an element of an array and an index.

- Its result is the element that is offset from the pointer by the index.
- Indices count from 0.
- The subscript operator returns an lvalue.

++ operator: When used with a pointer, the increment operator "adds one" by moving the pointer to refer to the next element in an array.

Example: ++i;

New and delete expressions

- Managing dynamic memory is error-prone
  - Memory leak when memory is not freed
  - Reading or writing to the object after is has been deleted
  - Applying delete to the same memory location twice

New and delete

new expression: Allocates dynamic memory.

- We allocate an array of n elements as follows: new type[n];
- new returns a pointer to the first element in the array.

delete expression: A delete expression frees memory that was allocated by new:

- delete [] p;
- p must be a pointer to the first element in a dynamically allocated array.
- The bracket pair is essential: It indicates to the compiler that the pointer points at an array, not at a single object.

New and Delete I

- **operator delete**: A library function that frees untyped, unconstructed memory allocated by operator new.
  - The library operator delete[] frees memory used to hold an array that was allocated by operator new[].
- **operator new**: A library function that allocates untyped, unconstructed memory of a given size.
  - The library function operator new[] allocates raw memory for arrays.
  - These library functions provide a more primitive allocation mechanism than the library allocator class.
  - Modern C++ programs should use the allocator classes rather than these library functions.

New and Delete II

- **member operators new and delete**: Class member functions that override the default memory allocation performed by the global library operator new and operator delete functions.
  - Both object (new) and array (new[]) forms of these functions may be defined.
  - The member new and delete functions are implicitly declared as static.
  - These operators allocate (de-allocate) memory. They are used automatically by new (delete) expressions, which handle object initialization and destruction.

New and Delete III

- **placement new expression**: The form of new that constructs its object in specified memory.
  - It does no allocation; instead, it takes an argument that specifies where the object should be constructed.
  - It is a lower-level analog of the behavior provided by the construct member of the allocator class.
- Example
  - allocator<string> alloc; string* sp = alloc.allocate(2); // allocate space to hold 2 strings
  - new (sp) string(begin,end) // construct a string from a pair of iterators

Copy control

copy control: Special members that control what happens when objects of class type are copied, assigned, and destroyed.

Rule of Three/Five: Rule of thumb: if a class needs a nontrivial destructor then it almost surely also needs to define its own copy constructor, an assignment operator and in the C++11 standard move constructor + move-assignment operator.

“resource acquisition is initialization” (RAII) idiom

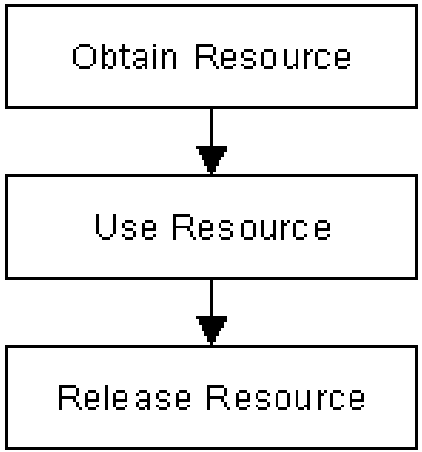
- In C++ RAII can be realized by smart pointers, e.g. the shared_ptr
-

```
class someResource {
    //internal representation holding pointers, handles etc.

public:
    someResource(){
        //Obtain resource.
    }

    ~someResource(){
    //Release resource.
    }

};
```



Standard Library allocator class

- Library class that allocates unconstructed memory
- If one uses new, memory is allocated and objects are constructed in that memory
- When allocating a block of memory, one often plans to construct objects in that memory when needed
- The allocator class allows to decouple construction from allocation

Managing Pointer Members

- Most C++ classes take one of the following approaches
  - The pointer member can be given normal pointerlike behavior. Such classes will have all pitfalls of pointers but will require no special copy control
  - The class can be given valuelike behavior. The object to which the pointer points will be unique and managed separately by each class object
  - The class can implement so-called smart pointer behavior. The object to which the pointer points is shared, but the class prevents dangling pointers

Pointer Members: Valuelike behavior

- **value semantics**: Description of the copy-control behavior of classes that mimic the way arithmetic types are copied.
  - Copies of valuelike objects are independent: Changes made to a copy have no effect on the original object.
  - A valuelike class that has a pointer member must define its own copy-control members.
  - The copy-control operations copy the object to which the pointer points.
  - Valuelike classes that contain only other valuelike classes or built-in types often can rely on the synthesized copy-control members.

Smart Pointers

- **smart pointer**: A class that behaves like a pointer but provides other functionality as well.
  - One common form of smart pointer takes a pointer to a dynamically allocated object and assumes responsibility for deleting that object.
  - The user allocates the object, but the smart pointer class deletes it.
  - Smart pointer classes require that the class implement the copy-control members to manage a pointer to the shared object.
  - That object is deleted only when the last smart pointer pointing to it is destroyed.
  - Use counting is the most popular way to implement smart pointer classes.

Use Count I

- **use count**: Programming technique used in copy-control members.
- A use count is stored along with a shared object.
- A separate class is created that points to the shared object and manages the use count.
- The constructors, other than the copy constructor, set the state of the shared object and set the use count to one.

Use Count II

- Each time a new copy is made—either in the copy constructor or the assignment operator—the use count is incremented.
- When an object is destroyed— either in the destructor or as the left-hand side of the assignment operator—the use count is decremented.
- The assignment operator and the destructor check whether the decremented use count has gone to zero and, if so, they destroy the object.

Standard Library smart pointers

- **shared_ptr**: allows multiple pointers to refer to the same object.
- Example
  - shared_ptr<int> pi(new int(1024));
- **unique_ptr**: owns the object to which it points.
- **weak_ptr**: does not control the lifetime of the object to which it points.