C and C+++ 5. C++ - Overloading - Namespaces - Classes Alan Mycroft University of Cambridge (heavily based on previous years' notes - thanks to Alastair Beresford and Andrew Moore) Michaelmas Term 2013-2014	C++ To quote Bjarne Stroustrup: "C++ is a general-purpose programming language with a bias towards systems programming that: a is a better C b supports data abstraction b supports object-oriented programming b supports object-oriented programming. C++ is "an (almost upwards-compatible) extension of C with support for: classes and objects (including multiple inheritance), operator overloading, exceptions and templates". [templates are a generalised form of generics] Much is familiar from Java, but with many subtle differences.
C++ fundamental types	 C++ enumeration Unlike C, C++ enumerations define a new type; for example enum flag {is_keyword=1, is_static=2, is_extern=4, }
 C++ has all the fundamental types C has character literals (e.g. 'a') are now of type char In addition, C++ defines a new fundamental type, bool A bool has two values: true and false When cast to an integer, true→1 and false→0 When casting from an integer, non-zero values become true and false otherwise 	 When defining storage for an instance of an enumeration, you use its name; for example: flag f = is_keyword Implicit type conversion is not allowed: f = 5; //wrong f = flag(5); //right The maximum valid value of an enumeration is the enumeration's largest value rounded up to the nearest larger binary power minus one The minimum valid value of an enumeration with no negative values is zero The minimum valid value of an enumeration with negative values is the nearest least negative binary power
 References C++ supports references, which provide an alternative name for a variable Generally used for specifying parameters to functions and return values as well as overloaded operators (more later) A reference is declared with the & operator; for example: int i[] = {1,2}; int & refi = i[0]; A reference must be initialised when it is defined A connection between a reference and what it refers to cannot be changed after initialisation; for example: refi++; //increments value referenced 	References in function arguments • When used as a function parameter, a referenced value is not copied; for example: void inc(int& i) { i++;} //bad style? • Declare a reference as const when no modification takes place • It can be noticeably more efficient to pass a large struct by reference • Implicit type conversion into a temporary takes place for a const reference but results in an error otherwise; for example: float fun1(float&); float fun2(const float&); void test() { double v=3.141592654; fun1(v); //Wrong fun2(v); Cf. Fortran call-by-reference
Overloaded functions	Scoping and overloading
 Functions doing different things should have different names It is possible (and sometimes sensible!) to define two functions with the same name Functions sharing a name must differ in argument types Type conversion is used to find the "best" match A best match may not always be possible: void f(double); 	Functions in different scopes are not overloaded; for example: void f(int); void example() { void f(double); void f(double);

- 4 f(1L); //f(long)
 5 f(1.0); //f(double)
 6 f(1); //Wrong: f(long(1)) or f(double(1)) ?

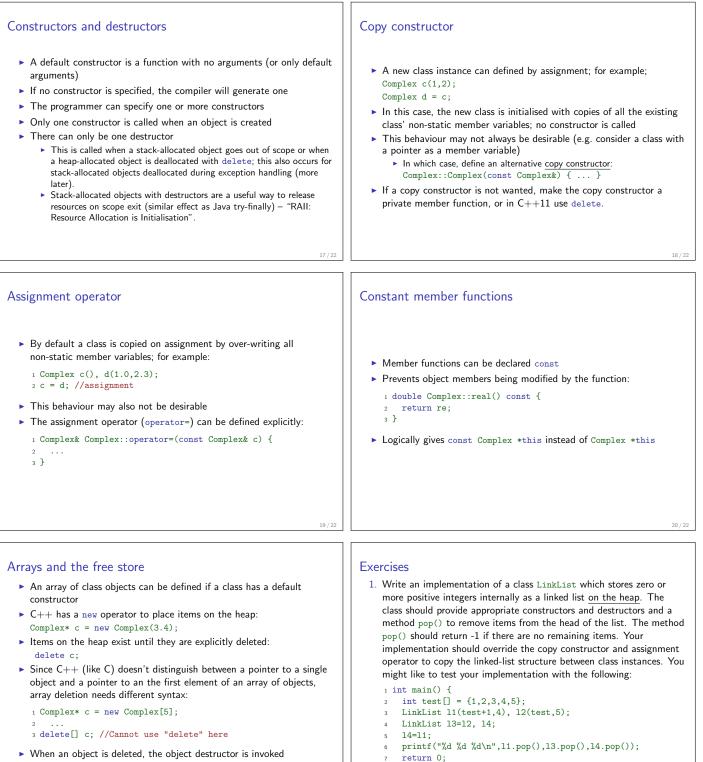
7/22



Example

User-defined types

► C++ provides a means of defining classes and instantiating objects 1 class Complex { Classes contain both data storage and functions which operate on double re,im; storage public: Classes have access control: Complex(double r=0.0L, double i=0.0L); private, protected and public 5 }: Classes are created with class or struct keywords 7 Complex::Complex(double r,double i) { struct members default to public access; class to private re=r,im=i; // deprecated initialisation-by-assignment A member function with the same name as a class is called a 9} constructor 10 A member function with the same name as the class, prefixed with a 11 int main() { 12 Complex c(2.0), d(), e(1,5.0L); tilde (~), is called a destructor return 0; 13 A constructor can be overloaded to provide multiple instantiation 14 } methods Can create static (i.e. per class) member variables 15/22 16/22



- ▶ When an array is deleted, the object destructor is invoked on each element
- 7
- 8 }

Hint: heap allocation & deallocation should occur exactly once!