Computer Languages

why all the fuss about C++

Kent Blackburn

Caltech /LIGO Project

June 3rd, 1997

The Problems

- Software costs are going up, and hardware costs are going down (NASA/EOS estimates $100 per line of code)
- Software development time is getting longer, and maintenance costs are getting higher, while at the same time hardware development time is getting shorting and less costly
- Software errors are becoming more frequent as hardware errors become almost nonexistent
- Software is developed using a rigidly structured process that is inflexible
- Changing conditions are making software obsolete long before delivery
- Only 25% of all software projects result in working systems

The Cost Numbers

Table 1: Software Project Costs by Development Phase

Software Proiect Phase	Percent of Proiect
Requirements	*3*
Design	*8*
Programming	*7*
Testing	*15*
Maintenance	*67*

Table 2: Cost of Correcting Software Errors

Software Development Phase:	Requirements Analysis	Design	Code and Unit Test	Integration Test	Validation and Documentation	Operational Maintenance
Development Funds	*5%*	*25%*	*10%*	*50%*	*10%*	
Errors Introduced	*55%*	*30%*		*10%*		*5%*
Errors Found	*18%*	*10%*		*50%*		*22%*
Relative Cost to Correct	*1x*	*1-1.5x*		*1-5x*		*10-100x*

- **Source**: Hughes Department of Defense Composite Software Error History

History of Programming

- **1960's:** small relatively simple applications/systems were developed with simple languages (assembly, FORTRAN, COBOL) by free spirited developers using the "*creative method*" resulting in "*spaghetti code*"
- **1970's:** method of "*structured analysis and design*" based on using the function as the building block came into being resulting in "*modular code*" that helped software developers manage the the functional organization of software but did very little to help developers manage the data
- **1980's:** software theory began to focus on the data management, implementing the "*entity*" and "*relational databases*" but these were found to be conversely inadequate at managing the function which lead to the development of "*object oriented programming*"
- **Today:** Object Oriented Programming views data and functions as equals, allowing developers to manage the complexities of software applications through a collaborative network of objects - when developers can manage more aspects of the problem they produce more flexible and maintainable software

Language Complexity

- The advantages of object oriented programming languages extract a major price - complexity of the language
- This leads to additional training requirements
- Introduces avenues for subtle misuses of the language
- C++ is intentionally constructed as an extension to C with similar syntax but very complex semantics
- C++ has evolved greatly as it matures to the now pending ANSI Standard

Table 3: Language Complexity of some of the more common computer languages

Language	Pascal	Modula-2	Modula-3	C	C++ v1	C++ v3	Ada	ANSI C++
Keywords	35	40	53	29	42	48	63	62
Statements	9	10	22	13	14	14	17	15
Operators	16	19	25	44	47	52	21	54
Ref. Man. Pages	28	25	50	40	66	155	241	650

- Source: AT&T Lucent Technologies

OOP Language Characteristics

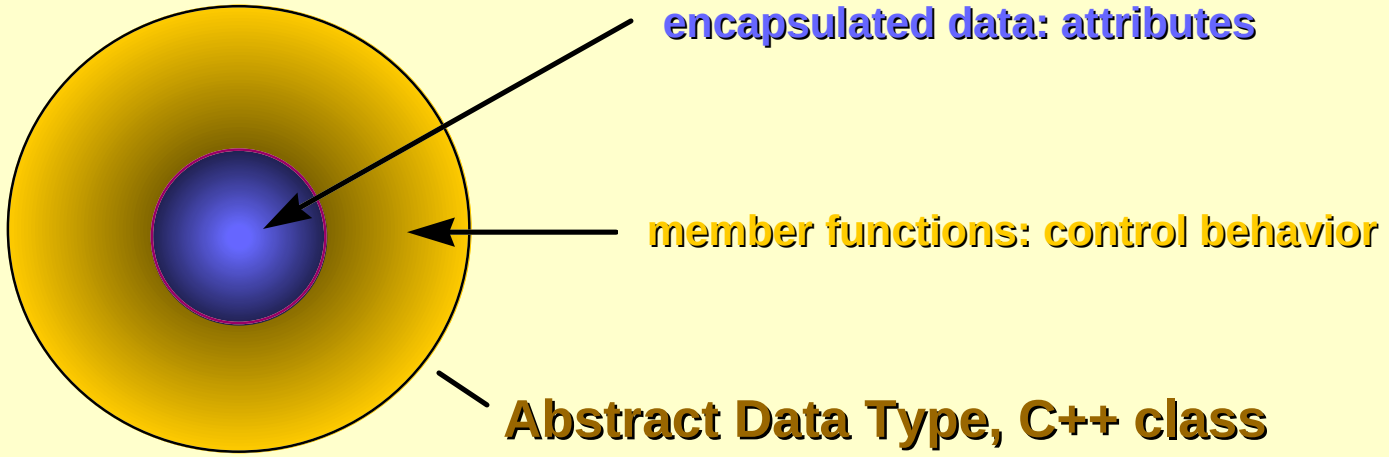
- **Object Oriented:** an adjective that is used to denote that a particular activity is done in a manner of thinking and organization that combines data and the processes which modify that data together into a single functional unit.
- **Encapsulation with data hiding:** The ability to distinguish an objects internal state and behavior from its external state and behavior
  - ***leads to more robust software***
- **Type-extensibility:** The ability to add user defined types(objects) to augment the native types fixed by the language
  - ***leads to software that more closely matches the real world***
- **Inheritance:** The ability to create new objects by importing or reusing the description of existing objects
  - ***leads to software reuse and greater maintainability***
- **Polymorphism with dynamic binding:** The ability of objects to be responsible for function invocations
  - ***leads to more extensible quality software system implemented at run time***
- **Exception handling:** The ability of a program to respond in a non fatal manor to error conditions
  - ***leads to clearer, more robust and more fault tolerant programs***

Capabilities of C++

- C++ is an advanced object oriented programming language supporting multiple inheritance, aggregation, and dynamic behavior
- C++ supports operator overloading to more naturally work with user defined types (classes)
- C++ is highly portable, now having an ANSI standard (draft)
- C++ is fast, not incurring the run-time expenses of type checking and garbage collection associated with most “pure” object oriented languages
- C++ does not require a graphical environment and is relatively inexpensive
- C++ is a marriage of low level assembly language and high level object oriented constructs, developers write code at the appropriate level to accurately model the problem
- C++ is a multi-paradigm language giving the developer a range of choices in design and coding solutions…object oriented programming is just one of the supported paradigms

Type-Extensibility in C++

- C++ supports type extensibility through “abstract data types”
- The language implemented data types such as int, float, double, struct, etc., are extended by user defined types called classes
- The C++ class is build up from simple data types in concert with member functions or “methods”, as well as user defined types

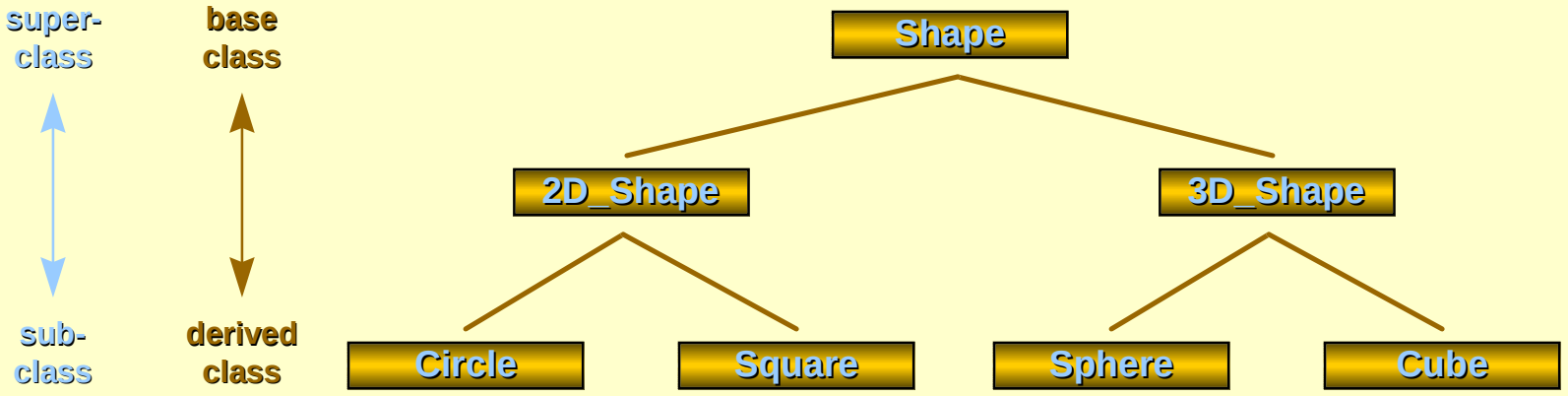


Encapsulation in C++

- C++ encapsulates data (attributes) and functions (behavior) into packages called objects
- The object is an instance of a class
- Objects communicate with each other through well defined interfaces without having to know the details of the implementation of other objects (this is information hiding)
- Attribute and behavior hiding is controlled by the three class member access specifiers: { public:, protected:, private: }
  - Public: specifies class members that can be seen by other objects
  - Protected: specifies class members that are hidden from unrelated objects, but access is granted to derived objects
  - Private: specifies class members that can't be accessed by any objects
- Special functions called "friend functions" can be granted access to protected and private members of an object

Inheritance in C++

- C++ supports inheritance through two distinct mechanisms
  - composition: objects are used as data members - (object_A *has an* object_B)
  - inheritance: objects are derived (subclassed) from base objects - (object_A *is an* object_B)
- Both mechanism motivate code reuse in an object oriented fashion
- Inheritance forms tree like hierarchical structure of class derivation



Polymorphism in C++

- C++ achieves polymorphism through the use of virtual functions in classes
- With polymorphism it is possible to design highly extensible systems
- Programs are written to generically process using the base class from a class hierarchy
- Example: A virtual print member function placed in the Shape class can be overloaded by all sub-classes so that at run-time the appropriate objects print method is invoked using dynamic binding
- C++ supports a second type of virtual function known as “pure” virtual functions which are used to create “abstract base classes”
- Abstract base classes (unlike concrete base classes) cannot be instantiated
- The sole purpose of abstract base classes is to provide appropriate base classes used to take full advantage of polymorphism
- Polymorphism and dynamic binding promotes extensible high quality code

Templates in C++

- Templates allow C++ to specify with a single code segment an entire range of related overloaded functions or classes
- An example would be a template sort function that needs to sort integers, floats, doubles, and character strings, one template function could be coded and the specialization to data type is handled by the compiler
- Templates are superior to the **#define** preprocessor directive for doing this since unlike macros, templates benefit from C++ type checking
- Templates promote code reuse since the function is coded only once for all data types
- Template classes are also used by C++ to perform type parameterizing
- Templates are related to inheritance as a means of deriving classes
- Templates are one of the more complex aspects of C++ programming to take full advantage of in a system due to the requirement for more detailed design development

Exception Handling in C++

- One of the latest additions to the C++ language is exception handling
- The extensibility of C++ can increase substantially the number of ways and the kinds of errors that can occur in software
- Through exception handling, clearer, more robust and more fault tolerant C++ code can be written
- C++ exception handling allows the programmer to locate error handling code away from the main lines of the code thus improving readability and modifiability of the code
- Exception handling is only designed to deal with “synchronous errors” such as an attempt to divide by zero, it cannot handle asynchronous situations such as mouse events and network message passing
- C++ exception handling using a “try - throw - catch” construct where a block of code is tried, if an exception occurs it is thrown and then caught following the try block

Pitfalls in C++

- C++ supports multiple programming paradigms: developing hybrid programming paradigms in a single software project undermines the advantages of any single approach and should be strongly avoided
- Quote: "***real men can program FORTRAN in any language***", this is all to true in C++ since it is not a "pure" object oriented language
- The more complex a language the greater the need for rules, developers need to adopt coding standards and practices as part of a C++ project
- There is a steep start-up curve associated with a successful C++ software project, developers must resist the temptation and pressures to shortcut the development process so important to OOP projects
- C++ developers need to remember to keep the size and number of ADT to a minimum in order to avoid unnecessary complexity in a particular project
- Because C++ is an extension to the C language it is important to become familiar with object oriented ways and avoid the fall-back to C syndrome

Recommended Reading

- With the ANSI C++ standard nearing completion, focus on references based on the draft standard and AT&T C++ version 3.0
- Focus on references that teach the object oriented paradigm and C++
- Avoid references that target specifics like *C++ for scientists* or *numerical methods in C++* as they skip over the necessary exposure to OOP
- Some of my personal favorites have been:
  - "*C++ Primer*" by Stanley B. Lippman, 2nd Edition, Addison Wesley
  - "*C++, How to Program*" by H. M. Deitel & P. J. Deitel, Prentice Hall
  - "*Object-Oriented Programming Using C++*" by Ira Pohl, 2nd Edition, Addison Wesley
  - "*UML and C++, a practical guide to object-oriented development*" by R. Lee & W. Tepfenhart, Prentice Hall
  - "Enough Rope to Shoot Yourself in the Foot, Rules for C and C++ programming" by Allen I. Holub, McGraw-Hill
  - "Ruminations on C++, a decade of programming insight and experience" by A. Koenig & B. Moo, Addison Wesley
  - "Collection and Container Classes in C++" by C. Hughes & T. Hughes, Wiley

Conclusion

- **Yes, C++ is a complicated language with a step learning curve**
- **Yes, C++ can produce more cost efficient and maintainable software**
- **Yes, physicists and engineers can learn to use C++ in their software applications and systems**
- **Yes, you can be productive using C++ in the first 6 months to 2 years**
- **Yes indeed, C++ is a very good thing!**
- **I hope you too will learn and benefit from the ways of OOP**