Elements of C and C++

Jim Fawcett
Summer 2017

Contents

C++ Goals

Contents

- "The language thus provided general mechanisms for organizing programs, rather than support for specific application areas. This was what made C with Classes - and later, C++ - a general-purpose language rather than a C variant with extensions to support specialized applications."

- "C with Classes was explicitly designed to allow better organization of programs; "computation" was considered a problem solved by C. I was very concerned that improved program structure was not achieved at the expense of run-time overhead compared to C."

- "I strongly felt then, as I still do, that there is no one right way of writing every program, and a language designer has no business trying to force programmers to use a particular style. The language designer does, on the other hand, have an obligation to encourage and support a variety of styles and practices that have proven effective and to provide language features and tools to help programmers avoid the well-known traps and pitfalls."

  Bjarne Stroustrup, "The Design and Evolution of C++"

  [underlines are mine]

Why Use a C Core?

Contents

- "C is clearly not the cleanest language ever designed nor the easiest to use so why do so many people use it?

  - ***C is flexible***: It is possible to apply C to most every applica-tion area and to use most every programming technique with C. The language has no inherent limitations that preclude particular kinds of programs from being written.
  - ***C is efficient***: The semantics of C are "low level"; that is , the fundamental concepts of C mirror the fundamental concepts of a traditional computer. Consequently, it is relatively easy for a compiler and/or programmer to efficiently utilize hard-ware resources for C programs.
  - ***C is available***: Given a computer, whether the tiniest micro or the largest super-computer, chances are that there is an acceptable quality C compiler available and that the C com-piler supports an acceptably complete and standard C lang-uage and library. Libraries and support tools are also avail-able, so that a programmer rarely needs to design a new system from scratch.
  - ***C is portable***: A C program is not automatically portable from one machine (and operating system) to another, nor is such a port necesarily easy to do. It is, however, usually possible and the level of difficulty is such that porting even major pieces of software with inherent machine dependencies is typically technically and econonmically feasible.

Compared with these first-order advantages, the second-order drawbacks like the curious C declarator syntax and the lack of safety of some language constructs become less important."

Bjarne Stroustrup, "The Design and Evolution of C++"

C Language Elements Contents

- Kernighan and Ritchie [1988] start this way:
  - C provides a variety of data types
    - characters, integers, and floating point numbers of several sizes
    - derived data types created with pointers, arrays, structures, and unions.
  - Expressions are formed from operators and operands
  - A very rich set of operators make for concise and expressive constructions.
  - Any expression can be a statement - simply append a semicolon
  - Pointers provide machine independent address arithmetic
  - C has control-flow constructs required for well structured programs:
    - statement grouping { ... }
    - decision making if-else
    - selecting on of several cases switch-case
    - looping with test at top while, for
    - looping with test at bottom do-while
    - skip to loop top continue
    - break out of loop break

And More C Elements Contents

- Paraphrasing Kernighan and Ritchie:

- Variables:
  - may be internal to a function, external but known only within a single source file, or visible to the entire program, e.g., local, file, or program scope.

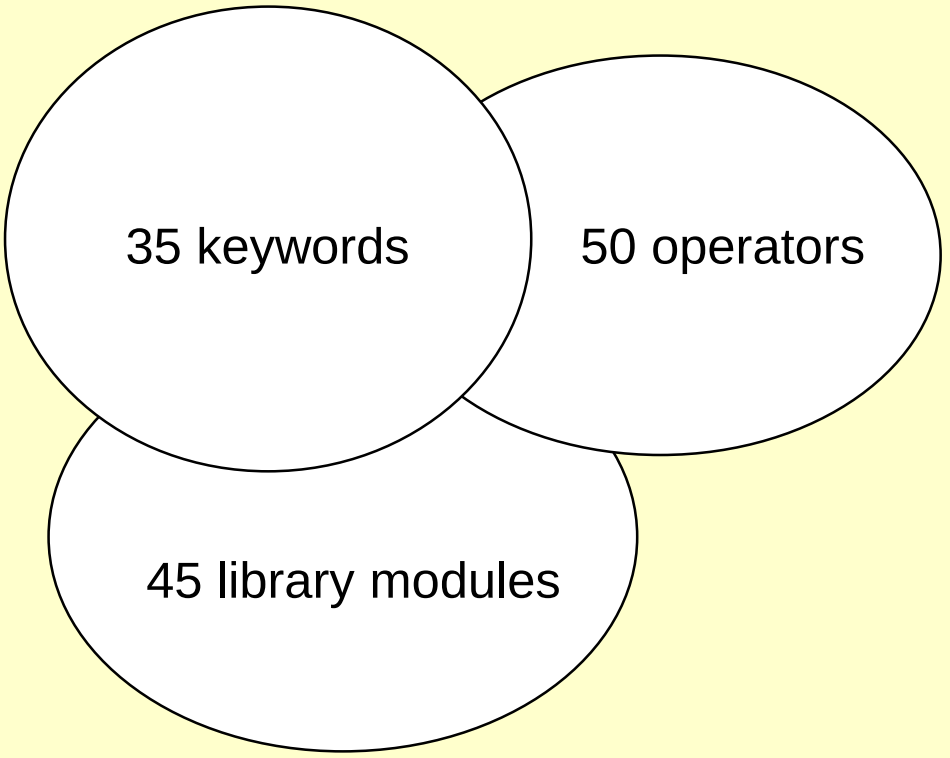
- Functions:
  - Function arguments are passed by value, minimizing side-effects, but that makes returning results from functions more complex
  - Any function may be called recursively
  - Functions may be passed and return values of basic types, structures, unions, or pointers
  - Local variables are created anew for each function invocation unless qualified as static
  - Function definitions may not be nested, but variables can be declared in a block-structured fashion.
    - blocks are delimited by { ... }
    - local declarations must appear at the beginning of each block before any expressions
  - The functions of a C program may exist in more than one source file and may be compiled separately.

More C Language Elements Contents

- There are three types of statements in a C program:
  - **preprocessor statements**
    - start with # and end with a newline
    - perform macro text substitution
    - provide text inclusion of other source files
    - define conditional compilations
  - **declarations and definitions** announce types for named variables and functions
    - appear at the beginning of a block and end with a semicolon
    - definitions reserve named memory locations
    - Only one definition is allowed for each variable and function
    - declarations simply announce types to the compiler. There may be multiple consistant declarations for a C program variable or function
  - **expressions** have values and are formed as combinations of operators and variables
    - the value of an expression is temporary, occupying anonymous storage, unless assigned to a named variable
    - temporaries are often call r-values
    - named variables are often called l-values

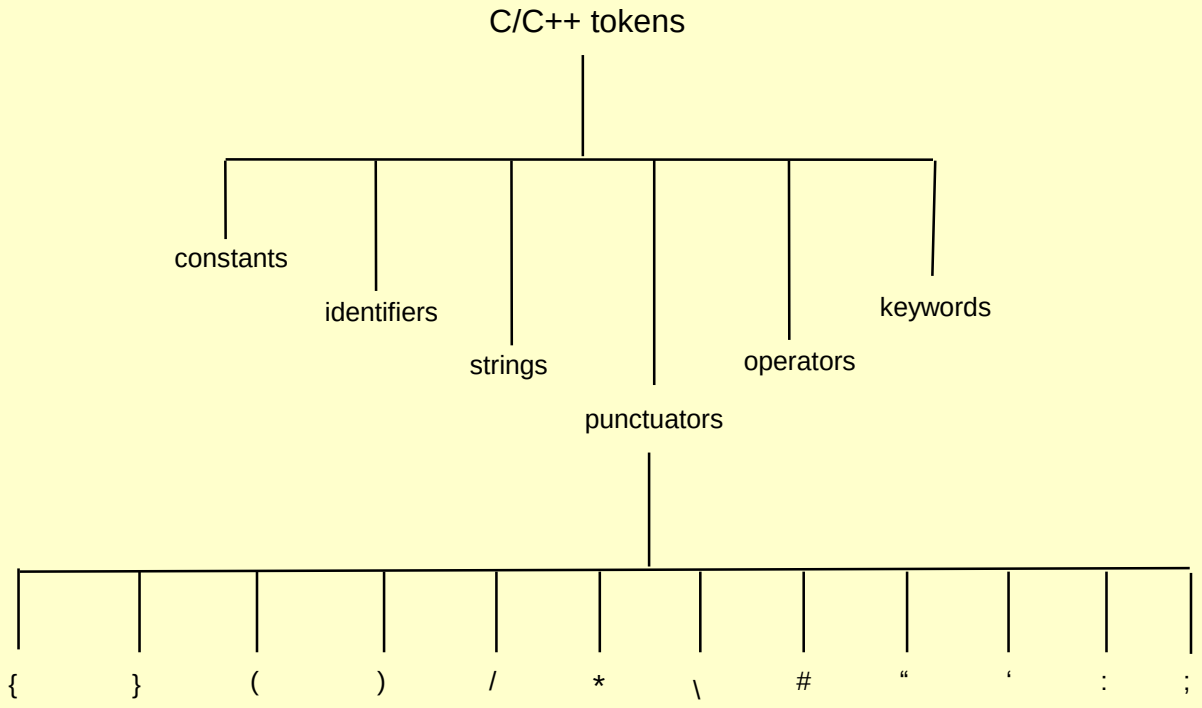
C Language at a Glance Contents

The C Language is essentially this:



C itself is simple, but has a rich, expressive set of operators and an extensive library of powerful components.

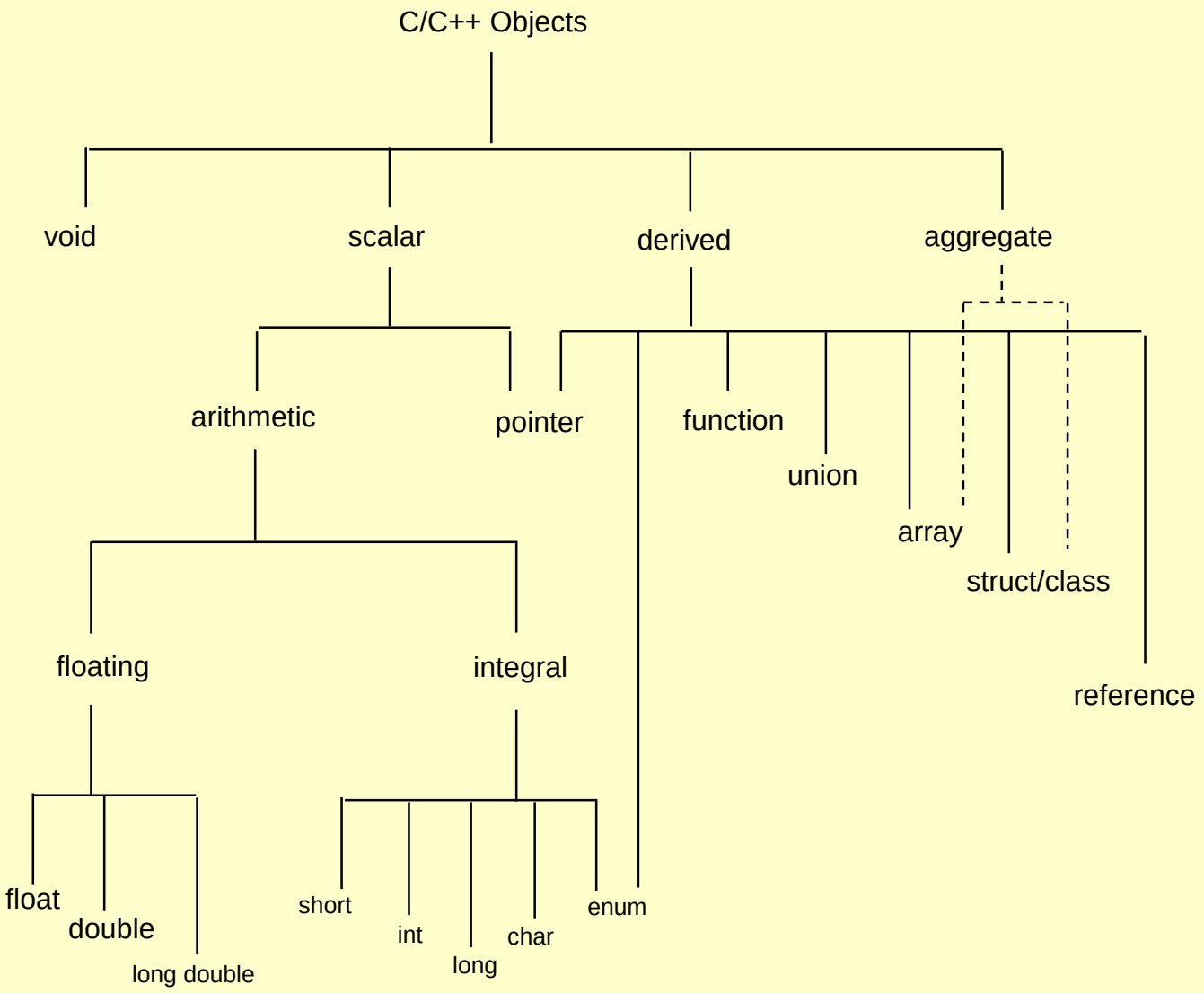
C/C++ Tokens Contents



{ } - group statements
( ) - group expressions to control order of evaluation
/* - begin comment
*/ - end comment
\ - line continuation
- begin preprocessor statement
“ - delimit beginning and end of string constant
‘ - delimit beginning and end of character constant
: - terminate class access controls public:, protected: and private:
; - terminate statements

C/C++ Objects

Contents



C/C++ Keywords

Contents

----< data definition >----------< C and C++ >----			
char	bool	double	- intrinsic data types
float	int	void	
const	extern	long	- data type modifiers
mutable	short	signed	
static	typename	unsigned	
enum	struct	union	- aggregate types

----< control flow >----------< C and C++ >----			
do	while	for	- looping constructs
break	continue		- loop short circuits
if	else		- logical branches
goto			- illogical branches
return			- function termination
switch	case	default	- case selection

----< class definition >----------< C++ only >----			
class			- user defined type
private	protected	public	- class access control
operator	virtual		- modifying functions
explicit			- prevent implicit convers.
this			- object state pointer
friend			- access override

C/C++ Keywords

Contents

----< user allocated memory management >----< C++ only >----

delete new

----< defining generic types >----------< C++ only >----

template export typename

----< handling exceptions >----------< C++ only >----

catch throw try

----< new style casts >----------< C++ only >----

const_cast static_cast dynamic_cast reinterpret_cast

----< run-time type information >----------< C++ only >----

typeid typeinfo - language defined type

----< system resource control >----------< C++ only >----

inline namespace using

----< system resource control >----------< C and C++ >----

asm auto register sizeof typedef volatile

C/C++ Operators

Contents

::	scope resolution	class_name::member	ltr
::	global	::name	
.	member selection	object.member	ltr
->	member selection	pointer->member	
[ ]	subscripting	pointer [int expr]	
( )	function call	expr (expr_list)	
( )	value construction	type (expr_list)	
sizeof	size of object	sizeof expr	
sizeof	size of type	sizeof (type)	
++	post increment	lvalue ++	rtl
++	pre increment	++ lvalue	
--	post decrement	lvalue --	
--	pre decrement	-- lvalue	
~	complement	~ expr	
!	not	! expr	
-	unary minus	- expr	
+	unary plus	+ expr	
&	address of	& lvalue	
*	dereference	* expr	
new	create (allocate)	new type	
new[ ]	create array	new type[int expr]	
delete	destroy (de-allocate)	delete pointer	
delete[ ]	destroy array	delete [ ] pointer	
( )	cast (type conversion)	(type) expr	
.*	member section	object.pointer-to-member	
->*	member section	pointer->pointer-to-member	
*	multiply	expr * expr	ltr
/	divide	expr / expr	
%	modulo (remainder)	expr % expr	
+	add	expr + expr	ltr
-	subtract	expr - expr	

C/C++ Operators

Contents

<<	shift left	expr << expr	ltr
>>	shift right	expr >> expr	
<	less than	expr < expr	ltr
<=	less than or equal to	expr <= expr	
>	greater than	expr > expr	
>=	greater than or equal to	expr >= expr	
==	equality test	expr == expr	ltr
!=	non equality test	expr != expr	
&	bitwise AND	expr & expr	ltr
^	bitwise exclusive OR	expr ^ expr	ltr
|	bitwise inclusive OR	expr | expr	ltr
&&	logical AND	expr && expr	ltr
||	logical inclusive OR	expr || expr	ltr
? :	conditional expression	expr ? expr : expr	rtl
=	assignment	lvalue = expr	rtl
*=	multiply and assign	lvalue *= expr	
/=	divide and assign	lvalue /= expr	
%=	modulo and assign	lvalue %= expr	
+=	add and assign	lvalue += expr	
-=	subtract and assign	lvalue -= expr	
<<=	shift left and assign	lvalue <<= expr	
>>=	shift right and assign	lvalue >>= expr	
&=	AND and assign	lvalue &= expr	
|=	OR and assign	lvalue |= expr	
^=	exclusive or and assign	lvalue ^= expr	
,	comma (sequencing)	expr, expr	ltr

The ANSI Standard C Library Contents

- The C language provides:
  - no operations to deal directly with composite objects such as character strings, lists, or arrays other than bit-for-bit assignment of structures
  - no storage other than static and local variables
  - no input and output to devices or files
- These capabilities and more are provided by the ANSI C library:
  - cstdio provides input and output functionality
  - cstdlib provides dynamic memory allocation
  - cstring has facilities for managing null terminated strings of characters
  - cctype provides support for testing a character's inclusion in classes, e.g., alphanumeric, whitespace, control, ...
  - cmath provides many of the elementary transcendental functions
  - cfloat helps in dealing with underflow, overflow and loss of precision in floating point operations

ANSI Standard C Library

Contents

- Additional C library capabilities are:
  - ctime // C-style date and time
  - cassert // macros supporting use of assertions
  - cerrno // C-style error handling
  - cctype // character classification
  - cwtype // classifying wide characters
  - cstring // management of C-style strings
  - cwchar // supports wide C-style strings
  - climits // numeric scalar limit macros
  - cstddef // C-language support
  - cstdarg // supports variable length argument lists
  - csignal // C-style signal handling

ANSI C++ Library Contents

- A standard for the C++ language has been approved by the ANSI X3J16 committee and ISO committee WG21. A draft standard was released for public review in February, 1994, and approved November, 1997. The official standard was approved in 1998.

- The standard defines a library which includes modules:
  - exception // defines exception, bad_exception
    stdexcept // defines all other exception types
    new // defines bad_alloc
    typeinfo // defines bad_cast, bad_typeid
    ios // defines ios_base::failure
  - new // augmenting operators new and delete
  - typeinfo // run-time type identification
  - iostream // io streams, standard stream objects
  - iomanip // formating, state control of iostreams
  - fstream // read/write to files you open by name
  - sstream // read/write char sequences in memory
  - string // create character string objects
  - complex // definition of complex numbers
  - numerics // numeric operations

- Here's a complete list:
  http://CppReference.com/w/cpp/header

ANSI C++ Library Contents

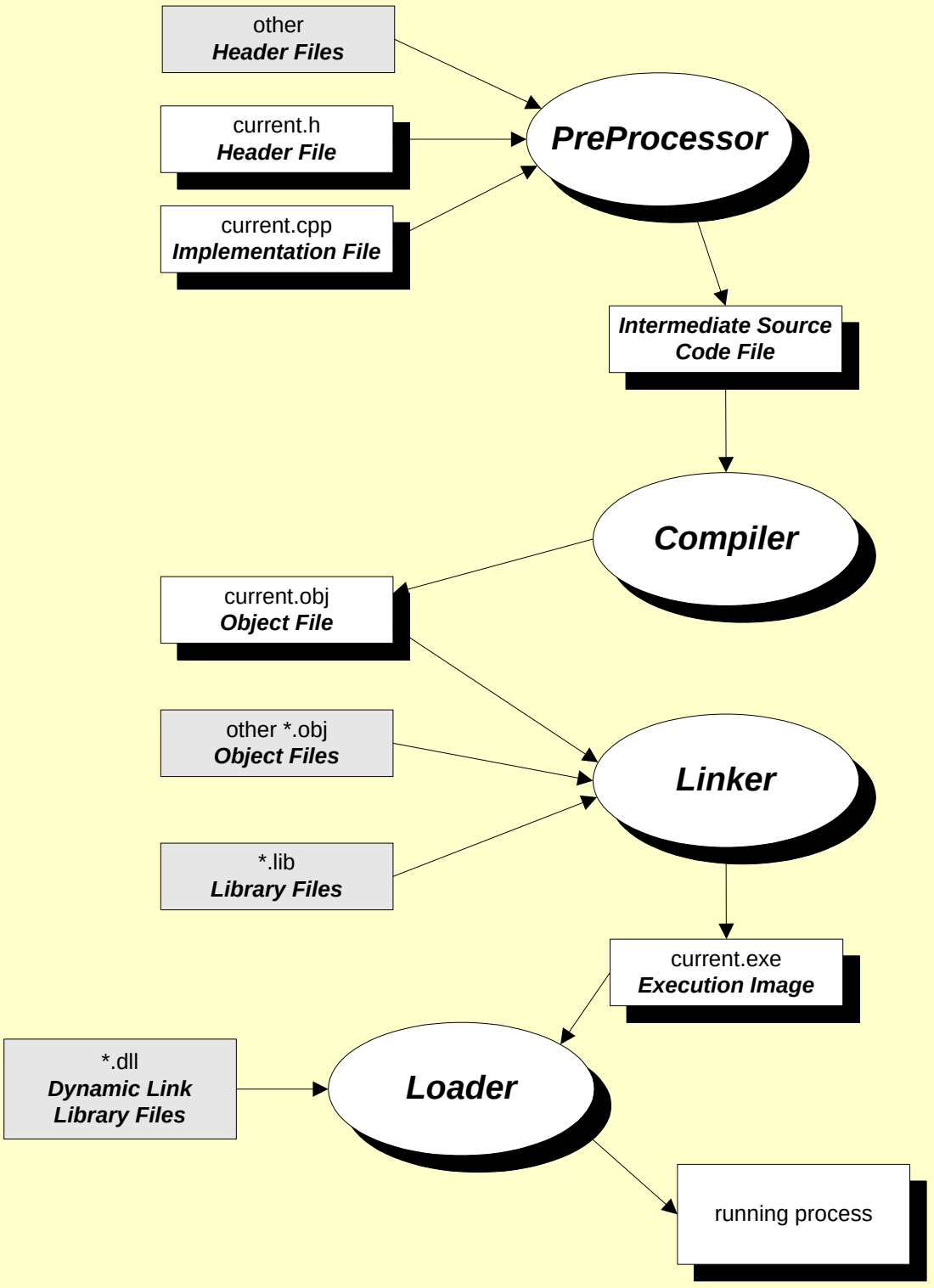
- A set of extensible generic programming modules are also provided, referred to collectively as the STL, or standard template library, which includes:

– functional	// function objects
– memory	// allocators for containers
– iterator	// support for iteration
– algorithm	// general algorithms
– vector	// expandable array
– list	// doubly linked lists
– deque	// double-ended queue
– queue	// queue of T
– stack	// stack of T
– map	// ordered set of pairs of // (key,value)
– set	// ordered set of key
– Unordered_map	// unordered set of pairs of // (key, value)
– bitset	// set of Booleans

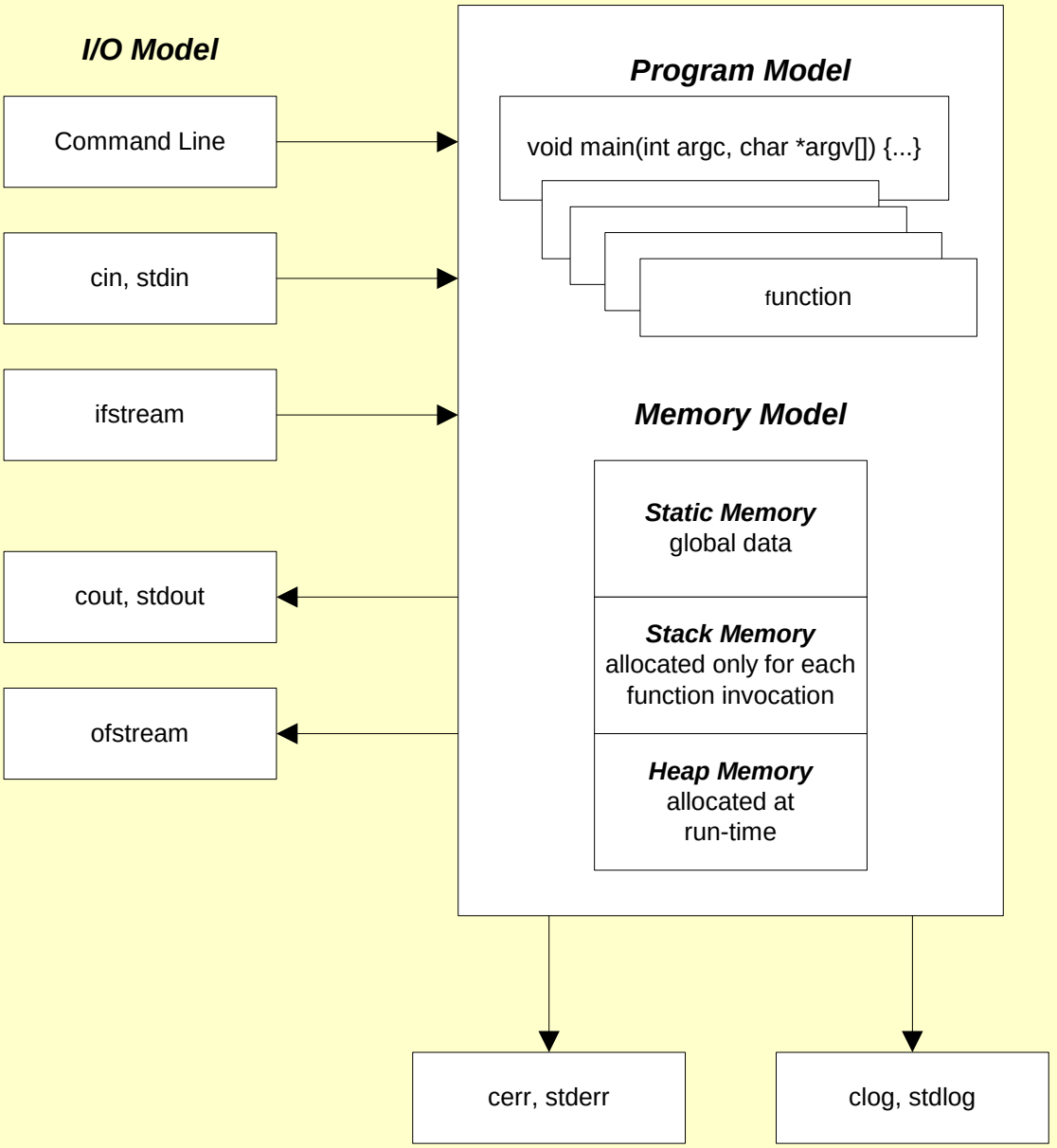
- Here's a complete list: http://CppReference.com/w/cpp/header

C/C++ Compilation Model

Contents



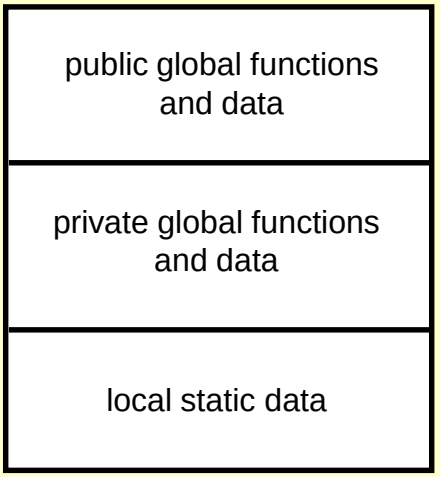
C/C++ Computational Model Contents



C/C++ Memory Model

Contents

Static memory: - available for the lifetime of the program

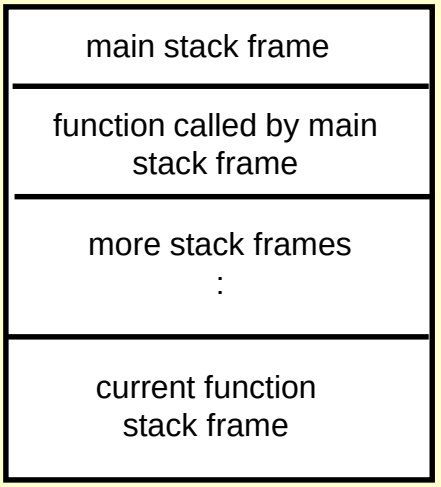


defined outside any function (globals) and initialized before main is entered.

global data and functions are made private by qualifying as static, otherwise they are public

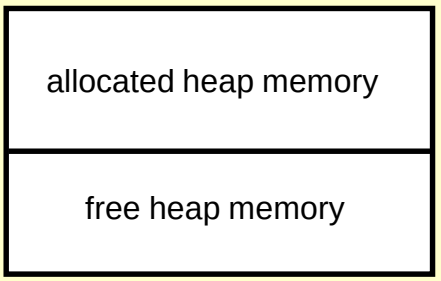
memory allocations local to a function, but qualified as static

Stack memory: - temporary scratch pad



- defined only while computational thread passes through a function.
- holds input parameters, local data, and return values, used as scratch-pad memory
- guaranteed to be valid during the evaluation of a containing expression, won't be valid after
- expression evaluation starts with function evaluation first, then expression evaluation as algebraic combination of terms
- stack frame is destroyed when expression evaluation is complete

heap memory: - valid from the time of allocation to deallocation



- allocated/deallocated at run time by invoking operators new /delete (or functions malloc/free)
- memory is available to anyone with a pointer to the allocated memory from the time of allocation until deallocated.

Run Time Flexibility

Contents

- **Compile-time memory allocation**
  Compile-time allocation of data objects is directly supported by the language.

```
int x[25];   /* size determined at compile-time */
```

  But frequently the designer does not know either the size of objects collected at run time or the number of objects that the program must handle.

- **dynamic memory allocation**.
  The Standard Library modules stdlib (C) and new (C++) provide simple and effective means of acquiring additional memory allocations from the operating system. These facilities are especially useful for creating composite data objects on demand on the heap at run-time.

```
X* x = (X*) malloc( numObjs * sizeof(X) );    (C)
X* x = new X[numObjs]                         (C++)
```

- **pointers**
  C provides access to dynamic memory through pointers. Address arithmetic supports incrementing and adding offsets to pointers. Arithmetic is "smart":
  - Incrementing a pointer adds to the pointer address the size of one object of the type pointed to.
  - Adding an integer, n, offset to a pointer creates an address n objects away from the original address.

Run-Time Flexibility Example Contents

- The standard containers are good examples of the flexibility C++ programs enjoy by using run-time allocation of resources - in this case memory from the heap.

- The str class (Handouts/CSE687/Code/STR) is a demonstration of how to build Abstract Data Types. Objects from the str class have a specific amount of character memory defined by the str constructor. If appends to the string build a character sequence which is longer than constructed memory size the str object will reallocate a larger memory block, copy over the current characters, and continue appending.

  None of this requires intervention by client code. The result of this is that str objects appear to have unlimited ability to append characters without wasting a lot of space when handling small strings.

- Advantages of run-time allocation are hard to over-state. Clients simply use str objects without needing a lot of fussy logic to make sure fixed memory allocations are not overrun.

  str use seems simple to clients, even though quite a bit of memory management is going on silently. That simplicity translates into fewer errors, and no anomalous behavior, as happens when fixed memory limits are reached.

Efficiency

Contents

- One goal of the C language was to replace the use of assembly language in system programming. The UNIX system, for example, is written mostly in C.
  - C was designed to create and manage composite data structures with small computational and memory costs
    - C is a strongly typed language that provides the designer with a lot of help in using complex data correctly
    - However, the language allows the designer to by-pass type matching through casts in order to support interfaces with hardware and other system software.
    - Furthermore, C provides a set of coercions between data types to flexibly support mixed mode expressions.
  - C does no run-time checking. It will:
    - support requests to index beyond static and dynamic array bounds for both reading and writing
    - maintain pointers to objects which go out of scope
    - copy data into memory that has never been allocated to the program or is no longer valid
    - attempt to return memory to the operating system more than once if asked to do so
  - Any of the preceding events spells **DISASTER**. The C language comes without seat belts. It will happily do what it is asked to do, assuming that the designer understands the consequenses of any request.

Efficiency Trade-offs in C++ Contents

- C has no run-time checking of array indexing, pointer arithmetic, or memory allocation status. The result is very fast programs, but the possibility of nasty errors induced by overrunning memory bounds, using uninitialized pointers, or invalid memory.

- C++ has the facilities to allow designers to buld in as much or as little run time checking as they want. It is even possible to build in checks that can be turned on or off as needed.

  An example is the str class. The indexing operator which returns values or writes into characters defined by s[i] has built in bounds checks on the index i. Should i be negative or exceed the allocated array size, the index operator will throw an exception with an informative error message.

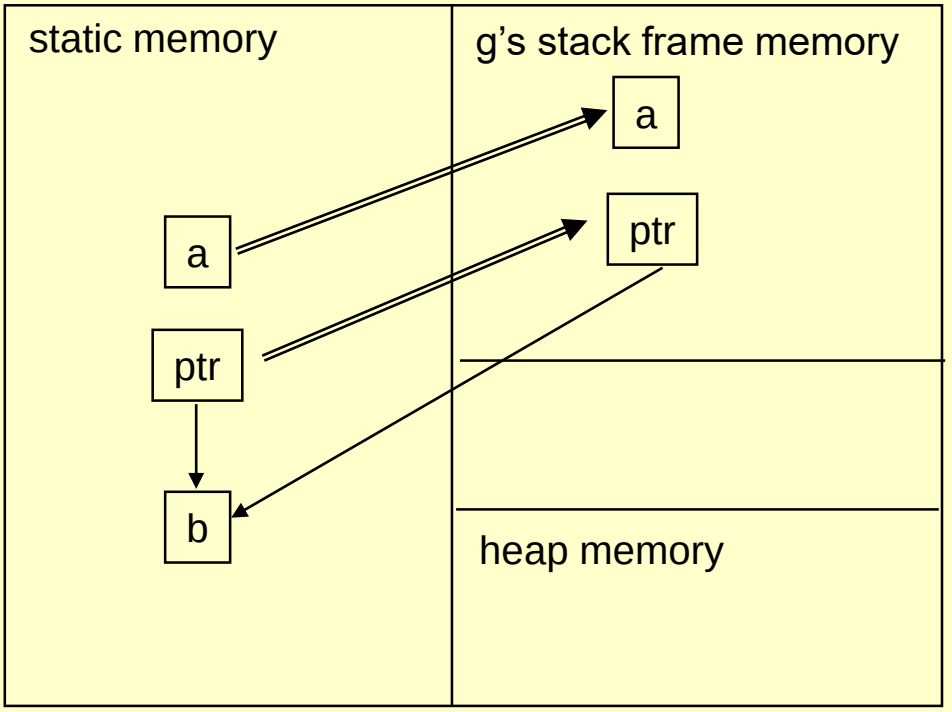
- This level of flexibility is just not available in the traditional languages like FORTRAN or Pascal.

- Note that, unlike my str class, bounds checking is not done in the standard C++ string class. This design choice was made for reasons of performance.

Elements of C++ Language Contents

- C++ is nearly a superset of C.
- Many, but not all, C programs will compile as C++ programs. C and C++ components are link compatible and so both may be parts of the same program.
- C++ strengthens C language support for procedural programming:
  - a new derived type, the reference, is provided to enable pass by reference, not provided by C.
  - Pass by reference makes external objects directly accessible in functions instead of providing copies of the original objects.
  - This allows the designer much more control over when side effects do or do not occur.
  - new input/output streams modules are provided which greatly simplify I/O.
  - C++ provides the keyword template to support the design of generic components - one design and implementation for many different types.
  - keyword inline supports the use of function syntax for inline code to eliminate function call overhead for very simple processing.
  - keywords catch and throw support the handling of exceptions

Pass by Value and by Reference Contents



- Suppose function g is declared as:

  void g(char a, char& b);

- When a is passed by value to function g(a,b) a copy of a is created on g's stack frame for all use by g. If g should change a's value, no change is made to the original back in static memory.

- when b is passed by reference to g(a,b) an implicit pointer to b is copied onto the stack frame for all accesses by g to the object b. Should g change the value of b it changes the original back in static memory.

- You should think of the reference parameter as an alias, e.g., another name, for the object passed to the function in that position.

More Elements of C++ Contents

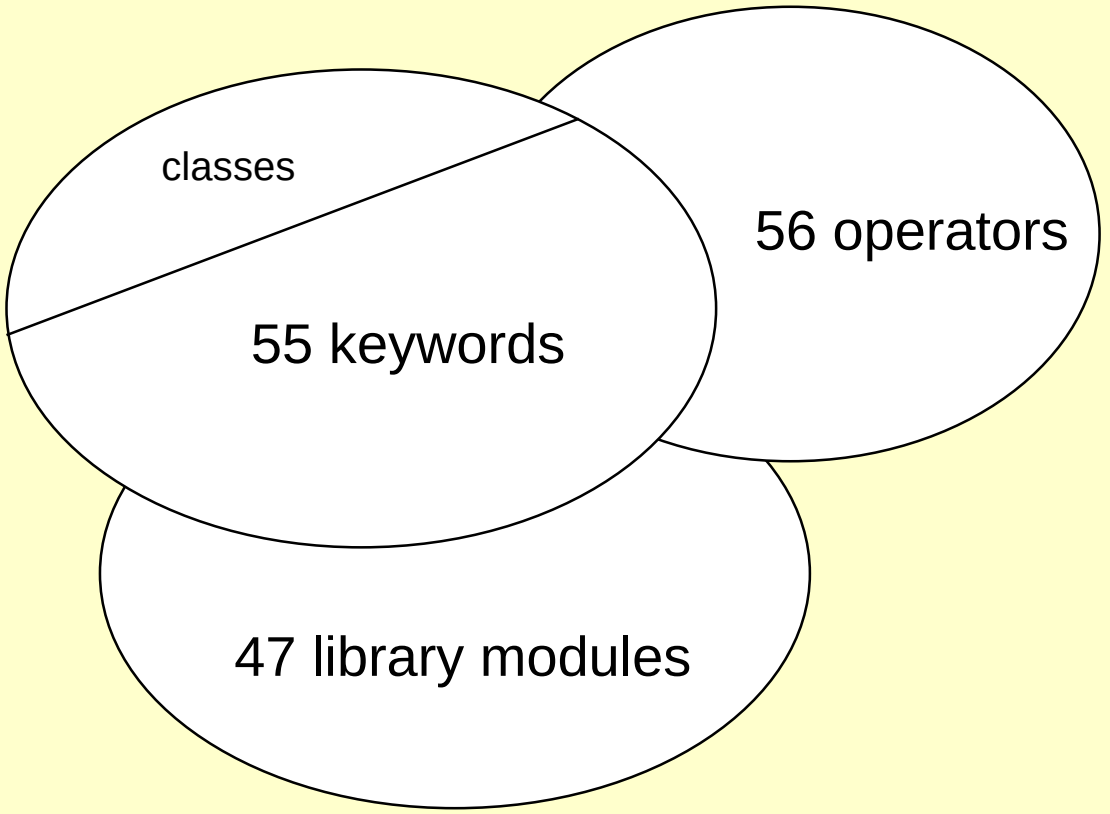
- C++ provides strong support for data abstraction:
- designers create new types using classes
  - classes have both data members and member functions
  - these are divided into a public interface and private or protected implementation
- objects (instances of a class) are essentially active data. Public members provide safe and simple access to data which may have complex internal and private management
- objects are declared and destroyed in exactly the same way that variables of the basic C language types are.
  - user defined constructors build class objects when they are declared
  - user defined destructors remove the objects when they go out of scope
- C++ operators new and delete directly support the run-time creation of objects
- Operators can be overloaded to have meanings unique to each class
  - overloading, which applies to all functions, not just operators, is accomplished by using the function's signature (name and types of formal parameters) as its identifier. Thus two functions with the same name but different argument tuypes represent unique functions.

Still More C++ Elements Contents

- C++ supports object oriented programming (OOP):
- OOP is concerned with building type relationships and type hierarchies
- objects may be composed by using one object as a data member of another object. This relationship is called aggregation or composition.
  - This is a very powerful way of building software incrementally.
- two classes which share some common attributes and behaviors can be derived from a common base class, sharing both design and code.
- public inheritance provides the derivation mechanism. It establishes an "is-a" relationship between derived classes and their base class. A derived object "is-a" base class object with extensions or modifications.
- since derived classes have all the attributes and behaviors of their base classes, client code will accept them wherever it expects a base class object.
  - this is a very powerful way of removing need-to-know from client code. Clients do not need to know the details which separate the various flavors of derived objects.
  - derived class objects tailor base class behaviors in a way suitable for their own needs by redefining virtual base class functions

C++ Language at a Glance Contents

The C++ language is essentially this:



Differences between C and C++ Contents

- C++ is almost a superset of the C programming language. There are many additions, and a few changes to the existing C syntax and semantics.

- The largest differences are introduction of classes, inheritance, and polymorphism to support object oriented software development and templates to support generic programming.

- A class represents a set of objects, all of which share the same attributes and behaviours, e.g., a user defined type.
  - Attributes are the set of values an object may assume. The specific values of an object's attributes are called it's state. The state of each object is unique.
  - Behaviours are all the operations allowable on an object's state. They are defined by class member and friend functions.

- Inheritance defines a relationship between classes.
  - public inheritance defines type compatibility relationships between the base and derived classes
  - private inheritance defines a uses relationship between base and derived classes.

- Polymorphism allows client code to treat all objects drawn from one inheritance hierarchy as members of the base class. Clients don't need to know all the details that distinguish one derived class from another.

Differences between C and C++ Contents

Additions

- Keywords ***class, friend, operator, private, protected, public, this***, and ***virtual*** support the structuring of classes
- Keywords ***new*** and ***delete*** support a revised dynamic memory management process
- Keywords ***template***, ***typename***, and ***export*** support the implementation of generic classes
- Keywords ***try***, ***catch*** and ***throw*** support exceptions.
- Keyword ***inline*** supports efficient partitioning
- Keywords ***const***, ***volatile***, and ***mutable*** allow the compiler to enforce rules concerning changes to program variables.
- keyword ***namespace*** supports large scale software development
- keyword ***typeid*** supports run-time type information
- keywords ***dynamic_cast***, ***static_cast***, ***const_cast***, and ***reinterpret_cast*** make casts visible and unambiguous

- **Additions** (continued)

  - ***Functions may be overloaded***. That is, one name may refer to more than one function. Overloaded functions are distinguished based on their signatures.

    The signature consists of a concatenation of the function name and type of each of its arguments. Return type is not part of the signature.

    The signature acts as an extended identifier and is bound to specific code at compile time.

  - A C++ expression can pass objects to a function by reference as well as by value (the C language supports only pass by value).

  - Function arguments can have default values. If the argument is omitted in an invocation, the default value is supplied.

  - Some simple functions can be inlined to avoid the overhead of function calls, e.g., creating a stack frame and copying parameters and return values.

- **Additions** (continued)

  - ***Virtual Member Functions may be overridden***. That is, one name may refer to more than one virtual function, based on whether it belongs to a base class or one of its derived classes. Overridden functions are distinguished based on the type of the object referred to by pointer or reference.

    The signature of overriding functions must match exactly with the signature used in the base class. There is one exception to this for covariant return types (see below).

    Since this is the C++ language, there is a qualification of this rule. If the return type of the base function is a base pointer or reference, the return type of the overriding function can be a pointer or reference to that derived type. This is called a covariant return type.

  - A client using a pointer or reference to an object of a class hierarchy (base-derived graph) the client does not need to know the type of object referred to. It simply uses the protocol provided by the base class public interface. Overridden (virtual) functions are called based on the object type, not on the type of the base class pointer or reference.

    This action is called polymorphism, and is a very powerful way of designing loosely coupled systems. Clients do not need to know anything about the design of a class hierarchy, nor are they bound in any way to those details. All they need to know is the base class protocol.

  - Very often polymorphic calls are the result of passing a function a base class pointer or reference, bound to a derived class object. This is one reason why C++ references are so important. These polymorphic calls are bound at run time.

Changes

- keyword static has additional semantics used to support the structuring of classes
  - original meaning #1:
    When qualifying a global function or variable, reduces visibility from linker public to file global.
  - original meaning #2:
    When qualifying a local variable, expands the valid lifetime from this invocation to the lifetime of the program.
  - new meaning #1:
    Static member data items in a class are shared among all objects of the class.
  - new meaning #2:
    Static member functions are invoked using the class name, as in C::memfunc(), and can operate only on static member data, global data, or formal parameters.

Obviously C++ is a context dependent language!

More Changes

- single line comments can now be created
  - the prefix string "//" starts a one line comment
  - traditional C style comments, /* ... */ are still supported
- C++ has added an "IOSTREAM" class implemented with standard library and header files which makes the ANSI C STDIO obsolete for simple I/O.

  Output to stdout is invoked by the statement:

```
int x = 5;
cout << x << "\n";
```

  which supplants the STDIO function:

```
int x = 5;
printf("%d\n",x);
```

- struct is optional when defining a data structure although still needed when declaring a structure

```
struct tag { ... };  // declaration
tag t;               // definition
```

- enum now has file scope unless declared extern

```
enum tag { name1, name2... } e1; // file scope
extern enum publicTag { nameA, nameB, ..} e2;
                                 // linker scope
```

Resources Used to Build C++ Programs

Contents

- 55 keywords
  - data definition
  - control flow
  - class definition
  - memory management
  - defining generic types
  - handling exceptions
  - managing type system (casts)
  - run-time type information
  - scope control
  - managing compiler information
- 56 operators
  - scope control
  - member and element access
  - arithmetic
  - logical operations
  - comparisons
  - memory management
- 47 standard C++ and C library modules
  - input and output
  - manipulation of character strings
  - mathematical operations
  - containers, iterators, algorithms
  - error management
  - Operating System interface

Learning and Reference Resources

- Sample Code
  - instructor's code
  - archive of your own homework and projects
- Class Texts
  - complete, accurate, effective references
- Class Notes
  - focus on concepts.
  - summary of language facilities, e.g., keywords, operators, libraries.
  - some detailed code examples in key areas - see chapters 4 and 5 and the appendices.
- Compiler
  - error messages
  - help system
- Instructor and Teaching Assistants