C. Glossary

`#define` | A `#define` (pronounced "pound define") is a mechanism to expand macros in a program. This macro expansion occurs before the program is compiled. The following example expands the macro `PI` into 3.1415 | Chapter 1.4

```
#define PI 3.1415
```

`#ifdef` | The `#ifdef` macro (pronounced "if-deaf") is a mechanism to conditionally include code in a program. If the condition is met (the referenced macro is defined), then the code is included. | Chapter 1.4

```
#ifdef DEBUG
   cout << "I was here!\n";
#endif
```

`#ifndef` | The `#ifndef` macro works exactly the same as `#ifdef` except code is included only when the condition is **not** met. | Chapter 1.4

```
#ifndef _FILE_H_
void function();
#endif
```

abstract class | An abstract class is a class containing at least one pure virtual function. This means that it is impossible to instantiate an object from an abstract class. | Chapter 3.5

abstracting | The process of creating an entity capturing the *essence* of relationships and allowing specific implementations to describe their details. Abstracting is the Object-Oriented way to addressing a problem involving class relations whereas universalizing is more of a procedural approach. | Chapte 3.0

access modifier | A part of the notation of the UML class diagram that allows the designer of the class to specify whether a given member variable or method is accessible to clients of the class or just to methods of the class itself. The three access modifiers are + for public, - for private, and # for protected. | Chapter 2.0 Chapter 2.2

accessor | Otherwise known as a "getter," an accessor is a method providing the client access to data from one or more member variable. | Chapter 2.3

```
int Date :: getYear() const
{
  return year;
}
```

address-of operator | The address-of operator (`&`) yields the address of a variable in memory. It is possible to use the address-of operator in front of any variable. | Review

```
{
   int variable;
   cout << "The address of 'variable' is "
        << &variable;
}
```

aggregate data type	A data type built from one or more built-in data types. An array is an aggregate data type because it consists of more than one instance of the base-type. Structures and classes are also aggregate data types	Chapter 1.2
algorithms	The part of a design document describing how certain functions are to be implemented. This is commonly expressed with pseudocode	Chapter 1.0
array	An array is a data-structure containing multiple instances of the same item. In other words, it is a "bucket of variables." Arrays have two properties: all instances in the collection are the same data type and each instance can be referenced by index (not by name). `{ int array[4]; // a list of four integers array[2] = 42; // the 3rd member of the list }`	Review
arrow operator	The arrow operator (`->`) enables the programmer to more easily access member variables and member functions from a pointer to a class or a pointer to a structure. The following are equivalent: `void display(const Point * pt) { cout << pt->x << (*pt).x; }`	Chapter 1.3
`assert`	An assert is a function that tests to see if a particular assumption is met. If the assumption is met, then no action is taken. If the assumption is not met, then an error message is thrown and the program is terminated. Asserts are designed to only throw in debug code. To turn off asserts for shipping code, compile with the `-DNDEBUG` switch.	Chapter 1.1
base class	A base class, also known as a parent class, is the class from which a derived class is built.	Chapter 3.2
binding	Binding is the process of connecting the name of a function or variable with the location in memory where the code or data resides. This may happen at compile time (called "early binding") or at run-time (called "late binding")	Chapter 3.1
bitwise operator	A bitwise operator is an operator that works on the individual bits of a value or a variable.	Review Chapter 2.6
`bool`	A `bool` is a built-in datatype used to describe logical data. The name "Bool" came from the father of logical data, George Boole. `bool isMarried = true;`	Review
Boolean operator	A Boolean operator is an operator that evaluates to a Bool (`true` or `false`). For example, consider the expression (`value1 == value2`). Regardless of the data type or value of `value1` and `value2`, the expression will always evaluate to `true` or `false`.	Review
buffer	An array of data to be filled with user input	Chapter 1.1

callback	A callback is a function pointer passed to another function as a parameter with the expectation that the function pointer will be executed.	Chapter 4.1
callee	A function that is called from another function. The function initiating the call is called the caller. The function receiving the call is the callee.	Chapter 4.1
caller	A function initiating a call to another function. The function receiving the call is the callee.	Chapter 4.1
casting	The process of converting one data type (such as a `float`) into another (such as an `int`). For example, `(float)3` equals `3.0`.	Review
`catch`	A part of the exception handling mechanism, the `catch` block receives an exception thrown by another part of the program `catch (int rocks)` `{` `cout << "Ouch!\n";` `}`	Chapter 1.2
`char`	A `char` is a built-in datatype used to describe a character or glyph. The name "Char" came from "Character," being the most common use. `char letterGrade = 'B';`	Review
comments	Comments are notes placed in a program not read by the compiler.	Review
cohesion	The measure of the internal strength of a module. In other words, how much a function does one thing and one thing only. The seven levels of cohesion are: Functional, Sequential, Communicational, Procedural, Temporal, Logical, and Coincidental.	Review
conditional expression	A conditional expression is a decision mechanism built into C++ allowing the programmer to choose between two expression, rather than two statements. `cout << (grade >= 60.0 ? "pass" : "fail");`	Review
coupling	Coupling is the measure of information interchange between functions. The four levels of coupling are: Data, Stamp, Control, and External.	Review
class	A data-structure consisting of member variables (like a structure) and member functions. A class is not a variable, but rather a template or plan by which a variable can be made. A variable built from a class is called an object.	Chapter 2.0
class diagram	One tool in the UML suite of tools, the class diagram helps the programmer design with classes. The following three row table is a class diagram for a simple class: `Name` `- first` `- last` `+ display`	Chapter 1.3 Chapter 2.0 Chapter 3.0
class template	A class template is similar to a function template with one exception: one or more member variable (rather than a parameter) uses an abstract data type. While function templates were useful for implementing generic	Chapter 4.3

	algorithms (like the bubble sort or the swap function), class templates are useful for implementing generic data-structures (like the stack or array).	
client	A function that is using code provided elsewhere in the program. If one function calls another function, the first is a client of the second. If one function instantiates an object from a class described elsewhere, the function is a client of the class.	Chapter 2.0
composition	A form of "has-a" where one class consists of a collection of subordinate classes.	Chapter 3.0
constructor	A constructor is a method in a class that is guaranteed to be called when an object is instantiated. There are three variations of constructors: default constructor (taking no parameters), a copy constructor (taking an object of the same type as the class itself as a constant by-reference parameter), and a non-default constructor (any other constructor that is not a default or copy constructor).	Chapter 2.4

```
Point::Point()
{
   x = 0;
   y = 0;
};
```

container	A container is data-structures designed to hold or contain data. Examples of containers are arrays, linked lists, the `vector` class, the stack, and the queue.	Chapter 4.6
`cout`	COUT stands for Console OUTput. Technically speaking, `cout` is the destination or output stream. In other words, it in the following example, it is the destination where the insertion operator (`<<`) is sending data to. In this case, that destination is the screen.	Review

```
cout << "Hello world!";
```

c-string	A c-string is how strings are stored in C++: An array of characters terminated with a null (`'\0'`) character.	Review
data type	A way of representing something in a computer program. Built-in data types include integers (`int`), real numbers (`float`, `double`, etc.), characters (`char`), etc. The programmer can create his own data type with structures (`struct`), enumerations (`enum`), typedefs (`typedef`), and more.	Chapter 1.3
data-structures	The part of a design document describing how data is stored in memory while the program is running. For an object-oriented program, the UML class diagram is commonly used here	Chapter 1.0
default parameter	Default parameters allow the author of a function to specify the value of a passed parameter if the client chose not provide one.	Chapter 1.5

```
void getline(char * text,             // required
             int bufferSize = 256);   // default
```

`delete`	The `delete` operator serves to free memory previously allocated with `new`. When a variable is declared on the stack such as a local variable, this is unnecessary; the operating system deletes the memory for the user. However, when data is allocated with new, it is the programmer's responsibility to delete his memory. `{` `int *pValue = new int;` `delete pValue;` `}`	Review
dereference operator	The dereference operator '*' will retrieve the data refered to by a pointer. `cout << "The data in the variable pValue is "` `<< *pValue;`	Review
derivation	A form of "is-a" class relations where one class is a manifestation or type of a parent class.	Chapter 3.0
derived class	A derived class, also known as the child class, is the class that is built from a base class.	Chapter 3.2
design document	A document containing a collection of tools used to help draft out the solution to a complex programming problem. Design documents ususally consist of the following components: problem definition, design overview, interface design, structure chart, algorithms, data-structures, file format, and error handling.	Chapter 1.0
design overview	The part of adesign document describing how the problem is to be solved. This is an overview of the big design decisions. Typically it consists of a high-level summary of the data-structures, algorithms, or function organization	Chapter 1.0
destructor	A method in a class that is guaranteed to be called when an object is destroyed such as when it falls out of scope. There is only one destructor; destructors cannot take parameters. `File::~File()` `{` `fout.close();` `}`	Chapter 2.4
`do … while`	One of the three types of loops, a DO-WHILE loop continues to execute as long as the condition in the Boolean expression evaluates to true. This is the same as a WHILE loop except the body of the loop is guaranteed to be executed at least once. `do` `cin >> gpa;` `while (gpa > 4.0 || gpa < 0.0);`	Review
`double`	A `double` is a built-in datatype use to describe large read numbers. The word "Double" comes from "Double-precision floating point number," indicating it is just like a `float` except it can represent a larger number more precisely. `double pi = 3.14159265359;`	Review

downcasting	Downcasting is the process of casting a base class into a derived class. This is dangerous because the program is required to "make up" information that is not provided.	Chapter 3.4

dynamically-allocated array	A dynamically-allocated array is an array that is created at run-time rather than at compile time. Stack arrays have a size known at compile time. Dynamically-allocated arrays, otherwise known as heap arrays, can be specified at run-time.
	`{`
	`   int *array = new int[size];`
	`}`
		Review

early binding	Early binding is the flavor of binding that the compiler performs. In a traditional program, the function name is "connected" with the code of the function at compile time. Errors in early binding are displayed as a compiler error.	Chapter 3.1
encapsulation	Encapsulation is the process of separating the use of an item from its implementation. This is one of the fundamental characteristics of OO programming: encapsulation, inheritance, polymorphism, and templates	Chapter 2.0

`eof`	When reading data from a file, one can detect if the end of the file is reached with the eof() function. Note that this will only return true if the end of file marker was reached in the last read.
	`if (fin.eof())`
	`   cout << "The end of the file was reached\n";`
		Review

error flags	An error handling technique where a program reports an error has occurred by returning `false`.	Chapter 1.2
error handling	The part of a design document describing the types of errors at may be encountered in the program, how to detect the errors, and what to do when the error condition arrises. There are three types of errors: user errors introduced by user input, file errors originating from unexpected data in a file, and internal errors resulting from bugs in the program	Chapter 1.0
error ID	An error ID (EID) is an error handling technique where all possible errors are enumerated. A function reporting the error would then return the code corresponding to the encountered error	Chapter 1.2
escape sequences	Escape sequences are simply indications to `cout` that the next character in the output stream is special. Some of the most common escape sequences are the newline (`\n`), the tab (`\t`), and the double-quote (`\"`)	Review
exceptions	A special error-handling mechanism built into the C++ language consisting of the `try`, `throw`, and `catch` elements	Chapter 1.2
expression	A collections of values and operations that, when evaluated, result in a single value. For example, `3 * value` is an expression. If value is defined as `float value = 1.5;`, then the expression evaluates to `4.5`.	Review

extraction operator	The extraction operator (>>) is the operator that goes between `cin` and the variable receiving the user input. In the following example, the extraction operator is after the `cin`. `cin >> data;`	Review Chapter 2.6
file format	The part of a design document describing how to persist that data between executions of the program	Chapter 1.0
`for`	One of the three types of loops, the FOR loop is designed for counting. It contains fields for the three components of most counting problems: where to start (the Initialization section), where the end (the Boolean expression), and what to change with every count (the Increment section). `for (int i = 0; i < num; i++)` `   cout << array[i] << endl;`	Review
`friend`	Friends are special functions in C++ that have access to the private member variables and private methods of a class.	Chapter 2.7
`fstream`	The `fstream` library contains tools enabling the programmer to read and write data to a file. The most important components of the `fstream` library are the `ifstream` and `ofstream` classes. `#include <fstream>`	Review
function pointer	A pointer to a function rather than a pointer to data. The following is a pointer to a void function taking an integer as a parameter: `{` `   void (*pointer)(int parameter);`	Chapter 1.5
function signature	A function signature is the combination of the name of the function with the data type of the parameters it takes. This is used to uniquely identify a funcition.	Chapter 1.5
function template	A function template is a mechanism in C++ allowing the programmer to implement a generic algorithm that works with any data type. `template <class T>` `void function(T & t)` `{` `}`	Chapter 4.2
`getline`	The `getline()` method works with `cin` to get a whole line of user input. `char text[256];          // getline needs a string` `cin.getline(text, 256);   // the size is a parameter`	Review
header file	A header file is a file containing function prototypes and type definitions. It is compiled through the `#include` mechanism.	Chapter 1.4

`ifstream`	The `ifstream` class is part of the `fstream` library, enabling the programmer to write data to a file. IFSTREAM is short for "Input File STREAM."
	
	`#include <fstream>`
	
	`{`
	`   ifstream fin("file.txt");`
	`   …`
	`}`
		Review

implementation file	Another name for a source file	Chapter 1.4
inheritance	A form of "is-a" class relations where one class is a manifestation or type of a parent class.	Chapter 3.0

inheritance indicator	An inheritance indicator is the part of the class definition where the programmer specifies which if any class is the base class. In the following example, `Base` is the base class and "`: public Base`" is the inheritance indicator.
	
	`class Derived : public Base`
	`{`
	… code removed for brevity …
	`};`	

`inline`	The `inline` keyword suggests to the compiler that it would be more efficient to copy-paste the code of the function body into the locations from which the function was called rather than call the function in the traditional manner.
	
	`inline void displayError()`
	`{`
	`   cout << "An error occurred\n";`
	`}`
		Chapter 1.5
		Chapter 2.3

insertion operator	The insertion operator (`<<`) is the operator that goes between `cout` and the data to be displayed. As we will learn in CS 165, the insertion operator is actually the function and `cout` is the destination of data. In the following example, the insertion operator is after the `cout`. `cout << "Hello world!";`	Review Chapter 2.6
interface design	The part of a design document describing the output, input, and various error messages of a program	Chapter 1.0
`int`	An `int` is a built-in datatype used to describe integral data. The word "Int" comes from "Integer" meaning "all whole numbers and their opposites." `int age = 19;`	Review
invariant	Once design consideration when working with "is-a" class relations, invariant attributes refer to those attributs that are the same between two classes.	Chapter 3.0

`iomanip` — The IOMANIP library contains the `setw()` method, enabling a C++ program to right-align numbers. The programmer can request the IOMANIP library by putting the following code in the program: — Review

```
#include <iomanip>
```

`iostream` — The IOSTREAM library contains `cin` and `cout`, enabling a simple C++ program to display text on the screen and gather input from the keyboard. The programmer can request IOSTREAM by putting the following code in the program: — Review

```
#include <iostream>
```

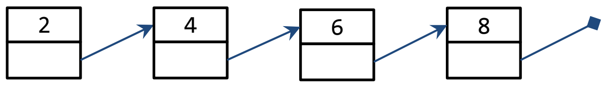
iterator — An iterator is a tool used to provide a standard way to iterate or traverse through a collection of values. They are designed to work the same regardless of the data type stored in the collection and regardless of how the collection is stored in memory. — Chapter 4.5

late binding — Late binding is the flavor of binding that occurs while the program is executing or running. Late binding errors are displayed as a run-time error such as a crash or a segmentation fault. — Chapter 3.1

```
Segmentation fault (core dumped)
```

All late binding mechanisms involve pointers, and all uses of pointers indicates late binding is at work. Therefore pointers to variables, arrays, and pointers-to-functions are indications of late binding.

linked list — A linked list is a type of data-structure where each item in the list is connected to the rest of the list through pointers. Each item in a linked list is called a node. — Chapter 4.4



local variable — A local variable is a variable defined in a function. The scope of the variable is limited to the bounds of the function. — Review

`makefile` — A `makefile` is a file describing how multiple files can be compiled together into a single program. It contains the dependencies for a given project. In the `makefile`, the programmer indicates how to tell if a given file needs to be rebuilt, and how. — Chapter 1.4

member variable — The list of variables comprising a structure or a class. In the following example, `row` and `col` are member variables — Chapter 1.3

```
struct Position
{
   int row;
   int col;
};
```

modulus — The remainder from division. Consider 14 ÷ 3. The answer is 4 with a remainder of 2. Thus fourteen modulus 3 equals 2: 14 % 3 == 2 — Review multi-dimensional array — A multi-dimensional array is an array of arrays. Instead of accessing each member with a single index, more than one index is required. The following is a multi-dimensional array representing a tic-tac-toe board: — Review

```
{
   char board[3][3];
}
```

mutator — Otherwise known as a "setter," a mutator is a method whose purpose is to modify or chang the data stored in one or more member variable. — Chapter 2.3

```
void Date :: setYear(int year)
{
   this-year = year;
}
```

`new` — It is possible to allocate a block of memory with the `new` operator. This serves to issue a request to the operating system for more memory. It works with single items as well as arrays. — Review

```
{
   int *pValue = new int;       // one integer
   int *array = new int[10];    // ten integers
}
```

node — A node is a structure or class consisting of some data-item and a pointer to the next node in a linked list. — Chapter 4.4

Node
data
pNext

null — The null character, also known as a null terminator, is a special character marking the end of a c-string. The null character is represented as `'\0'`, which is always defined as zero. — Review

```
{
   char nullCharacter = '\0';  // 0x00
}
```

`NULL` — The `NULL` address corresponds to the zero address `0x00000000`. This address is guaranteed to be invalid, making it a convenient address to assign to a pointer when the pointer variable does not point to anything. — Review

```
{
   int *pValue = NULL;   // points to nothing
}
```

object — A variable created from a class. In some situations, the term "object" also refers to a variable created from a structure — Chapter 2.2

object file — An object file is a partially compiled file. Rather than compiling a file directly into an executable (such as `a.out`), an object file is compiled into assembly which must be combined (or linked) with other object files before an executable is created. — Chapter 1.4

`ofstream`	The `ofstream` class is part of the `fstream` library, enabling the programmer to write data to a file. OFSTREAM is short for "Output File STREAM."
	`#include <fstream>`
	
	`{`
	`   ofstream fout("file.txt");`
	`   …`
	`}`
		Review

operator overloading	Operator overloading is the process of using a more convenient and human-readable notation for calling a function (such as "4 + 3"), than the functional notation common in programming languages (such as "`add(4, 3)`").	Chapter 2.6

overloading	Overloading is the process of having more than one function with the same name. The only differences between the functions are the parameters they take.
	`int   add(int   value1, int   value2);`
	`float add(float value1, float value2);`
		Chapter 1.5

pass-by-reference	Pass-by-reference, also known as "call-by-reference," is the process of sending a parameter to a function where the caller and the callee share the same variable. This means that changes made to the parameter in the callee will be reflected in the caller. You specify a pass-by-reference parameter with the ampersand `&`. `void passByReference(int &parameter);`	Review
pass-by-pointer	Pass-by-pointer, more accurately called "passing a pointer by reference," is the process of passing an address as a parameter to a function. This has much the same effect as pass-by-reference. `void passByPointer(int *pParameter);`	Review
pass-by-value	Pass-by-value, also known as "call-by-value," is the process of sending a parameter to a function where the caller and the callee have different versions of a variable. Data is sent one-way from the caller to the callee; no data is sent back to the caller through this mechanism. This is the default parameter passing convention in C++. `void passByValue(int parameter);`	Review
pointer	A pointer is a variable holding and address rather than data. A data variable, for example, may hold the value 3.14159. A pointer variable, on the other hand, will contain the address of some place in memory.	Review
polymorphism	Polymorphism is the process of one class having more than one variation. Each variation honors the same contract (called "interface") though the behavior details (called "implementation") may be different.	Chapter 3.1
`private`	An access modifier indicating that only member functions can have access to the corresponding member variable or method	Chapter 2.0 Chapter 2.2

problem description	The part of a design document describing what problem is to be solved. This commonly includes a high-level description of the functionality and behavior of the program	Chapter 1.0
`protected`	An access modifier indicating that only member functions and member functions of derived classes can have access to the corresponding member variable or method.	Chapter 3.3
prototype	A prototype is the name, parameter list, and return value of a function to be defined later in a file. The purpose of the prototype is to give the compiler "heads-up" as to which functions will be defined later in the file.	Review
pseudocode	Pseudocode is a high-level programming language designed to help people design programs. Though it has most of the elements of a language like C++, pseudocode cannot be compiled. An example of pseudocode is: `computeTithe(income)` / `RETURN income ÷ 10` / `END`	Chapter 1.0
`public`	An access modifier indicating that the client can have access to the corresponding member variable or method.	Chapter 2.0 Chapter 2.2
pure virtual function	A pure virtual function is a member function within a base-class that has no implementation. A class derived off of this base class, on the other hand, must have a definition for this function. This means that it is impossible to instantiate a class containing a pure virtual function (because there is no implementation for the pure virtual function) but it is legal to instantiate one of the derived classes.	Chapter 3.5
scope	Scope is the context in which a given variable is available for use. This extends from the point where the variable is defined to the next closing braces `}`.	Review
scope resolution operator	The scope resolution operator (`::`) is used to indicate to the compile which class the method is associated with: `int Card :: getSuit()` / `{` / `return value / 13;` / `}`	Chapter 2.2
separate compilation	The process of designing a program with multiple files, each of which is a manageable size.	Chapter 1.4
`sizeof`	The `sizeof` function returns the number of bytes that a given datatype or variable requires in memory. This function is unique because it is evaluated at compile-time where all other functions are evaluated at run-time. (see example below)	Review

```
computeTithe(income)
  RETURN income ÷ 10
END
```

```
int Card :: getSuit()
{
   return value / 13;
}
```

```
{
   int integerVariable;
   cout << sizeof(integerVariable) << endl;    // 4
   cout << sizeof(int)             << endl;    // 4
}
```

Appendix

slicing problem	The slicing problem is the process of casting a derived class into a base class. In this case, we are ignoring what is unique about the derived class and retaining only the base class information.	Chapter 3.4
source file	Also known as an implementation file, a source file is a file containing the source code for a program.	Chapter 1.4
stack variable	A stack variable, otherwise known as a local variable, is a variable that is created by the compiler when it falls into scope and destroyed when it falls out of scope. The compiler manages the creation and destruction of stack variables wherease the programmer manages the createion and destruction of dynamically allocated (heap) variables.	Review
STL	The Standard Template Library (STL) is a collection of tools designed to make common programming tasks easier.	Chapter 4.6

string	A "string" is a computer representation of text. The term "string" is short for "an alpha-numeric string of characters." This implies one of the most important characteristics of a string: is a sequence of characters. In C++, a string is defined as an array of characters terminated with a null character.
	`{`
	`   char text[256];   // a string of 255 characters`
	`}`
		Review

structure	A mechanism to create a custom data type based on previously defined data types
	`struct Position`
	`{`
	`   int row;`
	`   int col;`
	`};`
		Chapter 1.3

structure chart	A structure chart is a design tool representing the way functions call each other. It consists of three components: the name of the functions of a program, a line connecting functions indicating one function calls another, and the parameters that are passed between functions.	Chapter 1.0

structure tag	The part of a structure declaration indicating the name of the newly created data type. In the following example, Position is the structure tag
	`struct Position`
	`{`
	`   int row;`
	`   int col;`
	`};`
		Chapter 1.3

`static`	The static keyword is a modifier attached to a variable to indicate that only one copy of the variable will exist in a program
	`{`
	`   static int value;`
	`}`
		Chapter 2.5

`styleChecker`	`styleChecker` is a program that performs a first-pass check on a student's program to see if it conforms to the University style guide. The `styleChecker` should be run before every assignment submission.	Appendix A
tabs	The tab key on a traditional typewriter was invented to facilitate creating tabular data (hence the name). The tab character (`'\t'`) serves to move the cursor to the next tab stop. By default, that is the next 8 character increment. `cout << "\tTab";`	Review
TAR	TAR (Tape ARchiving utility, originally designed for creating an archive on the old-style reel-to-reel tape storage devices), is a program conceptually similar to ZIP: it combines multiple files into one and compresses them. We use TAR to combine our separate source files for the purpose of submission.	Chapter 1.4
template	A template is a function or class that can operate on a wide variety of data types, many of which do not have to be known by the developer of the function or class.	Chapter 4.0
template prefix	A statement immediately preceeding a function template declaration indicating the type of type parameters used in the function. `template <class T>`	Chapter 4.2
`this`	A pointer to the class itself from within the body of a method. This is useful to distinguish a member variable from a parameter of the same name.	Chapter 2.2 Chapter 2.8
`throw`	A part of the exception handling mechanism, the `throw` statement allows the programmer to signify that an error has occurred `throw "rocks";`	Chapter 1.2
throw list	A list of all the possible un-caught exceptions that a given function might throw. `void display() throw (bool);`	Chapter 1.2
`try`	A part of the exception handling mechanism, the try statement signifies that an exception may be thrown in the associated block of code `try { if (windows == true) throw "rocks"; }`	Chapter 1.2
type parameter	A data type associated with a template allowing a function or a class to be used with any data type. `{ T variable; }`	Chapter 4.2

UML	Unified Markup Language, UML is a suite of tools helping programmers design with classes and structures. The most commonly used UML tool and perhaps the most useful is the class diagram	Chapter 1.3 Chapter 2.0
universalizing	The process of making a single type containing all possible attributes. Universalizing is a more procedural approach to addressing a problem involving class relations whereas Abstracting is the Object-Oriented approach. Generally we should avoid universalizing.	Chapter 3.0
upcasting	Another name for the slicing problem, upcasting is the process of casting a derived class into a base class.	Chapter 3.4
variable	A variable is a named location where you store data. The name must be a legal C++ identifier (comprising of digits, letters, and the underscore _ but not starting with a digit) and conform to the University style guide (camelCase, descriptive, and usually a noun). The location is determined by the compiler, residing somewhere in memory.	Review
variant	Once design consideration when working with "is-a" class relations, variant attributes refer to those attributs that are different between two classes.	Chapter 3.0
virtual function	A virtual function is a member function that resides in the v-table associated with a class. This means that an object has a pointer to which version of a method is to be called.	Chapter 3.4
virtual function table	A virtual function table, also known as a v-table, is a structure containing function pointers to all the methods in a class. By adding a v-table as a member variable to a structure, the structure becomes a class	Chapter 2.1 Chapter 3.1
v-table	Another name for a virtual function table	Chapter 2.1
`void` pointer	A `void` pointer is a pointer to an unknown data type.	Chapter 4.1
`while`	One of the three types of loops, a WHILE-loop continues to execute as long as the condition inside the Boolean expression is true.	Review

```
while (grade < 70)
   grade = takeClassAgain();
```