Chapter 1

Snow Reflections

An Approach To The Art Of Programming

Learning Objectives

- *Identify and overcome the difficulties encountered by students when learning how to program*
- *List and explain the software development roles played by students*
- *List and explain the phases of the tight spiral software development methodology*
- *Employ the concept of the Flow to tap creative energy*
- *List and explain the primary areas of the Project Approach Strategy*
- *State the purpose of a header file*
- *State the purpose of an implementation file*
- *Explain the importance of separating interface from implementation*
- *Employ multi-file programming techniques to tame project complexity*
- *Explain the use of #ifndef, #define, and #endif preprocessor directives*
- *Apply preprocessor directives to implement multi-file programming projects*
- *State the importance of adopting consistent variable and constant naming conventions*
- *List and describe the two types of C++ comments*

Introduction

Programming is an art; there's no doubt about it. Good programmers are artists in every sense of the word. They are a creative bunch, although some would believe themselves otherwise out of modesty. Like any art you can learn the secrets of the craft. That is what this chapter is all about.

Perhaps the most prevalent personality trait I have noticed in good programmers is a knack for problem solving. Problem solving requires creativity, and lots of it. When you program a computer you are solving a problem with a machine. You transfer your knowledge of a particular problem into code, transform the code into a form understandable by a machine, and run the result on a machine. Doing this requires lots of creativity, especially when you find yourself stumped by a particular problem.

The material presented here is wrought from experience. Believe it or not, the hardest part about learning to program a computer, in any programming language, is not the learning of the language itself, rather, it is learning how to approach the art of problem solving with a computer. To this end the material in this chapter is aimed squarely at the beginner. However, I must issue a word of warning. If you are truly a novice, then some of what you read in this chapter will make less sense to you than to someone already familiar with C or C++. Do not worry, it is that way by design. If you feel like skipping parts of this chapter now, then go right ahead. The material will be here when you need it. In fact, you will grow to appreciate this chapter more as you gain experience as a programmer.

The Difficulties You Will Encounter Learning C++

During your studies of the C++ programming language you will face many challenges and frustrations. However, the biggest problem you will encounter is not the learning of the language itself, but the many other skills and tools you must learn before writing programs of any significance or gaining any measure of proficiency in solving problems with C++. If you are a seasoned student or practicing computer professional returning to the classroom to upgrade your skills, you have the advantage of experience. You can concentrate on learning the syntax and nuances of C++ and very quickly apply its powers to problems at hand. If you are an absolute beginner, however, you have much to learn.

Required Skills

In addition to the syntax and semantics of the C++ language you will need to master the following skills and tools:

- A development environment, which could be as simple as a text editor and compiler combination or a commercial product that integrates editing, compiling, and project management capabilities into one suite of tools,
- A computing platform of choice,
- Problem solving skills,
- How to approach a programming project,
- How to manage project complexity,
- How to put yourself in the mood to program,
- How to stimulate your creative abilities,
- Object-oriented analysis and design,
- Object-oriented programming principles.

The Planets Will Come Into Alignment

I use a metaphor to describe what it takes before you can get even the simplest program to execute properly. It is as if the planets must come into alignment. You must learn a little of each skill and tool listed above, with the exception of object-oriented programming principles and object-oriented analysis and design, to write, compile, and run your first C++ program. But, when the planets do come into alignment, and you see your first program compile and execute, and you begin to make sense of all the class notes, documentation, and text books you have studied up to that point, you will spring up from your chair and do a victory dance. It is a great feeling!

How This Chapter Will Help You

This chapter will give you the information you need to bring the planets into alignment sooner rather than later. It presents an abbreviated software development methodology that formalizes the three primary roles you play as a programming student. It will discuss some philosophical topics related to tapping into your creative energies. It will offer several strategies to help you manage project complexity, something you will not need for very small projects but should get into the habit of doing as soon as possible.

I recommend you read this chapter at least once in its entirety and refer back as necessary as you progress through the text and attempt increasingly difficult programming assignments.

Project Management

Three Software Development Roles

You will find yourself assuming the duties and responsibilities of three software development roles: Analyst, Architect, and Programmer.

Analyst

When you are handed a class programming project you may or may not understand what the instructor is actually asking you to program. Hey, it happens! Whatever the case may be, you, as the student, must read the assignment and design and implement a solution.

You can think of a project assignment as a requirements specification. They will come in several flavors. Some instructors go into painful detail about how they want the student to execute the project. Others prefer to generally describe the type of program they want thus leaving the details, and the creativity, up to you. There is no one correct method of writing a project assignment; each has its benefits and limitations.

A detailed assignment takes a lot of the guesswork out of what outcome the instructor expects. On the other hand, having every design decision made for you may prevent you from solving the problem in a unique, creative way.

A general project assignment delegates a lot of decision making to the student while also adding the responsibility of determining what project features will satisfy the assignment.

Both types of assignments model the real world to some extent. Sometimes requirements are well defined and there is little doubt what shape the final product will take and how it must perform. However, more often than not requirements are ill or vaguely defined. As an analyst you must clarify what is being asked of you. In an academic setting, do this by talking to the instructor and have them clarify the assignment. A clear understanding of the assignment will yield valuable insight into possible approaches to a solution.

Architect

Once you understand the assignment you must design a solution. If your project is extremely small you could perhaps skip this step with no problem. However, if your project contains several objects that interact with each other, then your design, and the foundation it sets, could make the difference between success and failure. A well-designed project reflects a subliminal quality that poorly designed projects do not.

Two objectives of good design are the ability to accommodate change and tame complexity. Change in this context means the ability to incrementally add features to your project as it grows without breaking the code you have already written. Several important object-oriented principles have been formulated to help tame complexity and will be discussed later in the book. For starters though, begin by imposing a good organization upon your source code files. You can use the source code file formats presented below to help in this endeavor.

Programmer

As programmer you will execute your design. The important thing to note here is that if you do a poor job as an architect your life as a programmer will be miserable. That doesn't mean the design has to be perfect. I will show you how to incrementally develop and make improvements to your design as you code.

Now that you know what roles you will play as a student let us discuss how you might approach a project.

A Project Approach Strategy

Most students have difficulty implementing their first significant programming assignment, not because they lack brains or talent, but because they lack experience. If you are a novice and feel overwhelmed by your first programming project rest assured you are not alone. The good news is that with practice, and some small victories, you will quickly gain proficiency at formulating approach strategies to your programming projects.

Even experienced programmers may not immediately know how to solve a problem or write a particular piece of code when tasked to do so. What they do know, however, is how to formulate a strategy to solve the problem.

You Have Been Handed A Project – Now What?

Until you gain experience and confidence in your programming abilities the biggest problem you will face when given a large programming assignment is where to begin. What you need to help you in this situation is a project approach strategy. The strategy is presented below and discussed in detail. I have also summarized the strategy in a checklist located in appendix A. Feel free to reproduce the checklist and use as required.

The project approach strategy is a collection of areas of concern to take into consideration when you begin a programming project. It is not a hard, fast list of steps you must take. It is intended to put you in control, to point you in the right direction, and give you food for thought. It is flexible. You will not have to consider every area of concern for every project. After you have used it a few times to get you started you may not ever use it explicitly again. As your programming experience grows feel free to tailor the project approach strategy to suit your needs.

Strategy Areas of Concern

The project approach strategy is formulated around areas of concern. These include requirements, problem domain, language features, and design. When you use the strategy to help you solve a programming problem your efforts become focused and organized rather than ad hoc and confused. You will feel like you are making real progress rather than drowning in a sea of confusion.

Requirements

A requirement is an assertion that specifies a particular aspect of expected behavior. A project's requirements are contained in a project specification or programming assignment. Ensure you completely understand the project specification. Seek clarification if you do not know, or if you are not sure, what problem the project specification is asking you to solve. In my academic career I have seen projects so badly written that I thought I had a comprehension problem. I'd read the thing over and over again until struck by a sudden flash of inspiration. But more often than not I would reinforce what I believed an instructor required by discussing the project with them.

Problem Domain

The problem domain is the specific problem you are tasked to solve. I would say that it is that body of knowledge necessary to implement a software solution apart and distinct from the knowledge of programming itself. For instance, "Write a program to simulate elevator usage in a skyscraper." You may understand what is being asked of you (requirements understanding) but not know anything about elevators, skyscrapers, or simulations (problem domain). You need to become enough of an expert in the problem domain you are solving so that you understand the issues involved.

Programming Language Features

The source of greatest frustration to novice students at this stage of the project is knowing what to design but not knowing enough of the language features to begin the design. This is when panic sets in and students begin to buy extra books in hopes of discovering the Holy Grail of project wisdom.

To save yourself from panic make a list of the language features you need to understand and study each one, marking them off as you go. This provides focus and a sense of progress. As you read about each feature, keep notes on their usage so you can refer to them when you sit down to formulate your program design.

Design

When you are ready to design a solution you will usually be forced to think along two completely different lines of thought: procedural vs. object-oriented.

Procedural Design

A procedural design approach is one in which you identify and implement program data structures separate from the functions that manipulate those data structures. When taking a procedural approach to a solution you will break the problem into small, easily solvable pieces, implement the solution to each of the pieces, and combine the solved pieces into a complete problem solution. The solvable pieces I refer to here are functions. This methodology is also known as functional decomposition.

Object-Oriented Design

Object-oriented design refers to designing with objects and their interfaces. Whereas a procedural design treats data structures separately from the functions that manipulate them, object-oriented design uses encapsulation to hide an object's implementation data structures behind a public interface. Data structures and the functions that manipulate them combine to form classes from which objects can then be created.

A problem solved with an object-oriented approach is decomposed into a set of objects and their behavior. Design tools such as the Unified Modeling Language (UML) can be used to help with this task. Once the objects in a system are identified, a set of interface functions is then identified for each object. Classes are declared and defined to implement the interface functions. Once all the program classes have been designed and written, they are combined and used together to form the final program. Note that when using the object-oriented approach you are still breaking a problem into solvable pieces, only now the solvable pieces are objects that represent the interrelated parts of a system.

Once you get the hang of object-oriented design you will never return to functional decomposition again. However, after having identified the objects in your program and the interfaces they should have, you will have to implement your design. This means writing class member functions one line of code at a time.

Think Abstractly

One mistake students often make is to think too literally. It is very important to remember that the act of solving a real world problem with a computer requires abstraction.

The Strategy In A Nutshell

Identify the problem, understand the problem, make a list of language features you need to study and check them off as you go. Once you formulate a solution to the problem, break the problem into manageable pieces, solve each piece of the problem, and then combine the solved pieces to form a total solution.

Applicability To The Real World

The programming problem solution strategy presented above is not intended to replace a formal course on software engineering, but it will help you when you enter the real world as a commercial programmer. In that world you will soon discover that all companies and projects are not created equal. Different companies have different design philosophies. Some companies have no software design philosophy. If you find yourself working for such a company you will probably be the software engineering expert!

The Art of Programming

Programming is an art. Ask any programmer and they will agree — it takes a lot of creativity to solve problems with a computer. Creative people have an advantage in that they are not afraid to explore new avenues of design. Their open-mindedness and readiness to accept new ideas give them the ability to see problems differently from people who tend toward the cut and dry. This section offers a few suggestions on how you can stimulate your creativity.

Don't Start At The Computer

Unless you have a good idea about what source code to write, sitting down at the computer first thing, without thinking through some design issues, is the worst mistake you can make. If you have ever suffered from writer's block when writing a paper for class then you can begin to understand what you will experience if you begin your project at the computer.

I recommend you forget the computer and go some place quiet and relaxing, with pen and paper, and draft a design document. It does not have to be big. Entire system designs can be sketched on the back of a napkin. The important thing is to have given some prior thought as to the design and structure of your program before you start coding.

The location you choose to relax in is important. It should be someplace where you feel really comfortable. If you like quiet spaces then seek quiet spaces; if you like to watch people walk by and think of the world, then an outdoor cafe may be the place for you. Inside, outside, at the beach, on the ski slope, wherever you prefer.

What you seek is the ability to let your mind grind away on the solution. Let your mind do the work. Writing code at the computer is a mechanical process. Formulating the solution is where real creativity is required, and is the part of the process that requires the most brainpower. Typing code is more like a drill on attention to detail.

Inspiration Strikes At The Weirdest Time

If you let your mind work on the problem it will offer its solution to you at the weirdest times. I solve most of my programming problems in my sleep. As a student I kept computers in the bedroom and would get up at all hours of the night to work on ideas that had popped into my head in a dream.

Try to have something to write on close at hand at all times. A pad of paper and pen next to the toilet comes in handy! You can also use a small tape recorder, or digital memo recorder, or your personal digital assistant. Whatever means suit your style. Just be prepared. There's nothing worse than the sinking feeling of having had the solution come to you in the middle of the night, or in the shower, or on the drive home from work or school, and say "I will remember that and write it down later," only to forget it and have no clue what you were thinking when you do finally get something with which to record your ideas.

Own Your Own Computer

Do not rely on the computer lab! I repeat. Do not rely on the computer lab! They are the worst places for inspiration and cranking out code. Most schools use PC's running Windows or some flavor of Unix and/or Macintosh computers.

You Either Have Time and No Money, or Money and No Time

The one good reason for not having your own personal computer to program your projects on is severe economic circumstance. Full-time students sometimes fall into this category. What they usually have gobs of is time. So much time that they spend their entire days at school and complain about not having a social life. But they can stay in the computer labs all day long and even be there when it is relatively quiet.

On the other hand, you may work full-time and be a part-time student. If this describes you then you don't have time to screw around driving to school to use the computer labs. You will gladly pay for any book or software package that makes your life easier and saves you time.

The Family Computer Is Not Going To Cut It!

If you are a family person working full-time and attending school part-time then time is a precious commodity. If you have a family computer that everyone shares, adults as well as children, then get another computer and put it off limits to everyone but yourself and password protect it. This will ensure your loving family does not accidentally wipe out your project the night before it is due through some unfortunate accident. It happens, don't kid yourself. Ensure your peace of mind by having your own computer in your own little space with a sign on it that reads "Touch This Computer And Die!"

Set The Mood

When you have a good idea on how to proceed with entering source code you will want to set the proper programming mood.

Location, Location, Location

Locate your computer work area someplace that's free from distraction. If you are single this may be easier than if you are married with children. If you live in a dorm or frat house good luck! Perhaps the computer lab is an alternative after all.

Have your own room if possible, or at least your own corner of a larger room that is recognized as a quiet zone. Noise canceling headphones might help if you find yourself in this situation.

Set rules. Let your friends and family know that when you are programming not to bother you. I know it sounds rude but when you get into the flow, which is discussed below, that is, if you ever get into the flow, you will be really upset when someone interrupts your train of thought to ask you about school lunch tomorrow or the location of the car keys. Establish the ground rules up front that say when it is a good time to disturb you when you are programming. The rule is - never!

Concept Of The Flow

Artists can really become absorbed in their work, not eating and ignoring personal hygiene for days, even weeks, at a time. Those who have experienced such periods of intense concentration and work describe it as a transcendental state where they have complete clarity of the finished product and tune out the world around them, living inside a cocoon of thought and energy.

Programmers can get into the flow. I have achieved the flow. You can achieve the flow and when you do you will crave the feeling of the flow again. It is a good feeling, one of complete and utter understanding of what you are doing and where you are going with your source code. You can do amazing amounts of programming while in the flow.

The Stages of Flow

Like sleep, there are stages to the flow.

Getting Situated

The first stage. You sit down at the computer and adjust your keyboard and stuff around you. Take a few deep breaths to help you relax. By now you should have a good idea of how to proceed with your coding. If not you shouldn't be sitting at the computer.

Restlessness

Second stage. You may find it difficult to clear your mind from the everyday thoughts that block your creativity and energy. Maybe you had a bad day at work, or a great day. Perhaps your spouse or significant other is being a complete jerk! Perhaps they're treating you very good and you are wondering why?

Close your eyes and breathe deep and regular. Clear your mind and think of nothing. It is hard to do but you can do it with practice. When you can clear your mind and free yourself from distracting thoughts you will find yourself ready to begin coding.

Settling In

Now, your mind is clear. Non-productive thoughts are tucked neatly away. You begin to program. Line by line your program takes shape. You settle in and the clarity of your purpose takes hold and propels you forward.

Calm and Complete Focus

You don't notice it at first, but at some point between this and the previous stage you have slipped into a deeply relaxed state and are utterly focused on the task at hand. It is like reading a book and becoming completely absorbed. Someone can call your name but you will not notice, and not respond until they either shout at you or do something to break your concentration.

You know you were in the flow, if only to a small degree, when being interrupted brings you out of this focused state and you feel agitated and have to settle in once again. If you avoid doing things like getting up from your chair for fear of breaking your concentration or losing your thought process then you are in the flow!

Be Extreme

Kent Beck, in his book "Extreme Programming Explained", describes the joy of doing really good programming. The following programming cycle is synthesized from his extreme programming philosophy.

The Programming Cycle

Plan

Plan a little. Your project design should serve as a guide in your programming efforts. Your design should also be flexible and accommodate change, which means that as you program, you may make changes to the design.

Essentially, you will want to design to the point where you have enough of the design to allow you to begin coding. The act of coding will soon reinforce your design decisions or detect fatal flaws that you must correct if you hope to have a polished, finished project.

Code

Code a little. Write code in small, cohesive modules. A class or function at a time is good granularity.

Test

Test a lot. Test each class, module or function separately or in whatever grouping makes sense. You will find yourself writing little programs on the side called test cases to test the code you have written. It is a good practice to get into. A test case is nothing more than a small little program you write and execute in order to test the functionality of some component or feature you have finished coding before integrating that component or feature into your project. The objective of testing is to break your code and correct its flaws before it has a chance to break your project in ways that are hard to detect.

Integrate

Integrate often. Once you have a tested module of code, be it either a function or complete set of related classes, integrate these tested components into your project regularly. The objective of regular integration is to see if the newly integrated components break any previously integrated components. If they do then remove them from the project and fix the problem. If a newly integrated component does break something you may have discovered a design flaw or a previously unnoticed dependency between components. If this is the case then the next step in the programming cycle should be performed.

Factor

Factor the design when possible. If you discover design flaws or ways to improve the design of your project you should factor the design to accommodate further development. An example of design factoring might be the migration of common elements from derived classes into the base class to take better advantage of code reuse.

Repeat

Apply the programming cycle in a tight spiral fashion. You will quickly reach a point in your project where it all starts to come together, and very quickly so.

The Programming Cycle Summarized

Plan a little, code a little, test a lot, integrate often, factor the design when possible. **Don't Wait Until You Think You Are Finished Coding The Entire Project To Compile**! Trying to write the entire program before compiling a single line of code is the most frequent mistake new programmers tend to make. The best advice I can offer is don't do it! Use the programming cycle outlined above. Nothing will depress you more than seeing a million compiler errors scroll up the screen.

A Helpful Trick: Stubbing

Stubbing is a programmer's trick you can use to speed development and avoid having to write a ton of code just to get something useful to compile. Stubbing is best illustrated by example.

Say that your project requires you to display a text-based menu of program features on the screen. The user would then choose one of the menu items and press enter, thereby invoking that menu choice. What you would really like to do is write and test the menu display and choice functions without worrying about actually performing the indicated action. You can do exactly that with stubbing.

A stubbed function is a function that exists to display a simple message to the screen saying in effect "Yep, the program works great up to this point. If it were actually implemented you'd be using this feature right now!"

Stubbing is a great way to incrementally develop your project. Stubbing will change your life!

Fix The First Compiler Error First

OK. You compile some source code and it results in a slew of compiler errors. What should you do? I recommend you stay calm, take a deep breath, and fix the first compiler error first. Not the easiest compiler error, but the first compiler error. The reason is because the first error detected by the compiler, if fatal, will generate other compiler errors. Fix the first one first and you will generally find a lot of the other errors will also be resolved. If you pick an error from the middle of the pack and fix it, you may introduce more errors into your source code! Fix the first error first!

Managing Project Complexity

Large projects differ from small projects in many ways. Large projects have more of everything: more variables, more user-defined types, more functions, more lines of code, and more complexity. There are two types of complexity: conceptual complexity and physical complexity.

Try to imagine a lot of something, like the number of dump truck loads required to move Mount Everest to North Carolina. Imagining large numbers poses a certain amount of conceptual complexity. Large software projects are very conceptually complex and many such projects end in failure because the conceptual complexity became impossible to manage. Object-oriented analysis and design (OOAD) techniques were developed to help tame conceptual complexity.

Conceptual complexity is accompanied by physical complexity. Large software development projects usually have many people working on many parts of the code at the same time. To ensure success, software developers adopt development standards. Development standards are rules developers must follow to ensure other developers can understand their work. Development standards may address issues like file naming, file location, configuration management, commenting requirements, and many, many other smart things to do to tame physical complexity.

Later in the book you will be taught how to tame conceptual complexity. This section presents you with a few smart things to do to help you manage the physical complexity of your projects. No project is too small to benefit from the techniques presented below. It is a good idea to develop good project management habits early in your programming career.

A word of warning: You could ignore the advice given here and manage to get small, simple projects to run, but if you try and structure large projects like small, simple projects, cramming all your code into one long file, you will doom yourself to failure. Formulate good programming habits now. Bad programming habits are hard to break and will end up breaking you in the long run.

Split Even Simple Projects Into Multiple Source Code Files

One of the first programming skills you must learn to help manage physical complexity is how to create multiple file projects. Your simplest programming project will have three files: a header file, an implementation file, and a main file. Larger projects will have more. As a rule of thumb you will have one header file and one implementation file for each class or abstract data type you declare. There will be only one main file which contains the main() function.

I will discuss these files and what goes into each one in more detail below, but first, I want to tell you why you want to learn the skill of developing multi-file projects. The following discussion about class interfaces may be somewhat advanced for novice readers. Fear not! Classes are discussed in great detail later in the book.

Separating a Class's Interface from its Implementation

When you design a system using object-oriented techniques you model the system's functionality by identifying objects within the system and how they interact with each other. Each object will have a certain behavior associated with it, along with an interface that allows other objects to access that behavior.

An object will belong to a class of objects. A class of objects is modeled in C++ using the struct or class construct. When you declare a new user-defined type representing an object in the system you are modeling you will create a new class. In this class you will declare a set of public methods. It is this set of public methods that become the interface to objects of that class. Because the class declaration contains the prototypes for the public class interface functions, and therefore considered as the interface to class objects, you will put class declarations in header files. I will talk more about header files below.

After you have declared the interface to a class of objects you need to define class behavior. You define class behavior by implementing the class member functions you declared in the class declaration. All class member functions declared for a class will be defined in a separate implementation file. I will talk more about implementation files below too.

If all this talk of classes, objects, and interfaces makes little or no sense to you now, just hang in there. It is all covered in much greater detail later in the book.

Benefits of Separating Interface from Implementation

You reap many benefits by declaring a class in one file and defining its behavior in another. I will talk about a few of those benefits now.

Makes Large Project File Management Easier

The larger the project, the more source code files it will contain. Putting each class declaration into its own header file and its implementation in a separate implementation file allows you to adopt a simple file naming convention. Namely, name the file the same name as the class it contains suffixed by either an "h", meaning header, or "cpp", meaning C++ implementation file. Giving your files the same names as the classes they contain makes finding them among tens, hundreds, or even thousands of files a heck of a lot easier.

Increases Portability

Portability refers to the ability of source code to be ported to another computer system. Although seamless portability is difficult to achieve without serious prior planning, you can make it easier to achieve by keeping platform or operating system dependent code separate. Putting class declarations and implementations in separate files helps you do just that.

Allows You to Create Class Libraries

Putting class declarations and implementations in different files will let you create class libraries. With a class library you can share the interface to your class or classes along with the compiled implementation code. You keep the C++ source code to the implementation and thereby protect your rights to your hard work.

Helpful Preprocessor Directives

Before compiling your source code, a C++ compiler will preprocess your code. It does this by invoking a program called the preprocessor. The preprocessor performs macro substitution, conditional compilation and filename inclusion. You tell the preprocessor what to do by putting preprocessor directives in your source code.

While there are many different preprocessor directives available for your use, you need only learn four of them to help you create and manage multiple file projects and thus help you manage the physical complexity of your projects. These are #ifndef, #define, #endif, and #include. As your C++ expertise grows you will find many other uses for these directives, as well as uses for other preprocessor directives not covered in this section.

#ifndef, #define, #endif

You can use this combination of preprocessor directives together to help you perform conditional compilation of your header files or source code. The purpose of using these three directives in your header file is to prevent the header file and its contents from being included multiple times in a project. The reason multiple header file inclusion is not a good thing is because a header file will contain function and/or data type declarations. A function or data type declaration should be made only once in a program. Multiple declarations make compilers unhappy!

The best way to illustrate their usage is by example. The C++ source code shown in example 1.1 represents a small header file called test.h that declares one function prototype named test().

1.1 test.h

```
#ifndef TEST_H
#define TEST_H

void test();

#endif
```

The #ifndef directive stands for “if not defined”. It is followed by an identifier, in this case TEST_H. The #define directive means exactly that, “define”. It is followed by the same identifier. The #endif directive stands for “end if”. It signals the end of the #ifndef preprocessor directive. The body of the header file appears between the #ifndef and #endif directives. This includes the #define directive and the function prototype test().

Remember that the purpose of the preprocessor directives is to communicate with the C++ preprocessor. What will happen in this case is the preprocessor will encounter the #ifndef directive and its accompanying identifier. If the identifier TEST_H has not been previously defined then the #define directive will be executed next, defining TEST_H, followed by the declaration of test().

On the other hand, if TEST_H has been previously defined, then everything between the #ifndef and #endif will be ignored by the preprocessor.

#include

Use the #include directive to perform file inclusion. There are essentially two ways to use the #include directive: #include <*filename*> and #include “*filename*”. Substitute the name of the header file you wish to include for the word *filename*.

The first usage, #include <*filename*>, will instruct the preprocessor to search in a number of directory locations as defined in your development environment. Most development environments let you customize this search sequence. If found, the entire #include line is replaced with the contents of *filename*.

The second usage, #include "*filename*", acts much like the first with the usual difference of checking first for *filename* in a user default directory. If *filename* is not found in the user's default directory then the preprocessor searches a list of predefined search locations.

The Final Word on Preprocessor Directive Behavior

The behavior of many C++ language features is implementation dependent, meaning the exact behavior is left up to the compiler writer. The search paths of the #include directives will be different for each development environment. To learn where your compiler is searching for header files and more importantly, how to make if find your header files when you create them, consult your compiler documentation.

Project File Format

Your projects will be comprised of many header and implementation files and one main file. This section shows you the general format of each file and what goes into each one. I will use the declaration of a simple class as an example.

Header File

Example 1.2 represents the contents of a file named firstclass.h

1.2 firstclass.h

```
#ifndef FIRSTCLASS_H
#define FIRSTCLASS_H

class FirstClass{
 public:
   FirstClass();
   virtual ~FirstClass();

 private:
   static int object_count;
};
#endif
```

Several conventions used here are worth noting. First, the name of the header file, firstclass.h, reflects the name of the class declaration it contains in lowercase letters with the suffix "h". Second, the identifier FIRSTCLASS_H is capitalized. The name of the identifier is the name of the file with the "." replaced with the underscore character "_". Doing these two simple little things makes your programming life easier by making it easy to locate your class header files and taking the guesswork out of generating identifier names for the #ifndef and #define statements.

Header files can contain other stuff besides class declarations. The following table will prove invaluable in helping you remember what you should and shouldn't put in header files.

Header Files Can Contain...	Examples
Comments	// C++-style comments /* C-style comments */
Include Directives	# include <helloworld.h> #include "helloworld.h"
Macro Definitions	#define ARRAY_SIZE 100

Table 1-1: Header File Contents

Header Files Can Contain...	Examples
Conditional Compilation Directives	#ifndef FIRSTCLASS_H
Name Declarations	class FirstClass;
Enumerations	enum PenState {up, down};
Constant Definitions	const int ARRAY_SIZE = 100;
Data Declarations	extern int count;
Inline Function Definitions	inline static int getObjectCount(){ return object_count; }
Function Declarations	extern float getPay();
Template Declarations	template<class T> class MyClass;
Template Definitions	template<class T> class MyClass{ };
Type Definitions	class MyClass{ };
Named Namespaces	namespace MyNameSpace{ }

Table 1-1: Header File Contents

It is just as helpful to know what you should not put in a header file. The following table offers some advice.

Header Should Not Contain...	Examples
Ordinary Function Definitions	float getPay() {return itsPay; }
Data Definition	double d;
Aggregate Definitions	int my_array[ ] = { 3, 2, 1};
Unnamed Namespaces	namespace { }
Exported Template Definitions	export template<class T> setVal(T t) { }

Table 1-2: What Not To Put In A Header File

Implementation File

Now that FirstClass is declared in firstclass.h definitions must be given for each of the member functions. In this case there are two functions to define, the constructor, FirstClass() and the destructor ~FirstClass(). C++ implementation files are suffixed with "cpp". Name the implementation file the same name as the header file and add the "cpp" suffix to the filename. Thus, the implementation file for FirstClass is named firstclass.cpp. The code for firstclass.cpp is given in example 1.3.

Main File

The main file is a C++ implementation file but instead of defining class member functions it contains the main() function. It has the same suffix, "cpp", as any other implementation file. I recommend naming this file main.cpp. This makes finding your main file an easy task.

1.3 firstclass.cpp

```
#include “firstclass.h”
#include <iostreams>

/*********************************
 Initialize classwide static variables first
*********************************
*/
int object_count = 0;

/*********************************
 Define member functions
*********************************
*/
FirstClass::FirstClass(){
 object_count ++;
 cout<<“There is/are: “ <<object_count
     <<“ FirstClass object(s)!”<<endl;
}

FirstClass::~FirstClass(){
 if((--object_count) == 0)
    cout<<“Destroyed last FirstClass object!”<<endl;
    else
        cout<<“There are: “<<object_count
            <<“ FirstClass objects left!”<<endl;
}
```

1.4 main.cpp

```
#include “firstclass.h”

int main(){
 FirstClass f1, f2, f3;
}
```

That’s it! Main files, and the main() function, should be kept short.

Commenting

A well commented program will be easier to understand by not only yourself but by others who read your code as well. There are two ways to comment source code. The first way involves adding additional, explicit comment lines to your source code by way of comment delimiters of which there are two styles: C and C++. The second way to comment your code is to write self-commenting code. This may sound complicated but it is easy to do. Besides making your code easier to read, writing self-commenting code reduces the need to rely on the first way of commenting. It also increases code reliability because you will find problems with your code easier if your code is easy to read and understand.

C-Style Comments

Add C-style comments to your code by enclosing text between two sets of delimiters: “/*” and “*/”. For example:

1.5 C-style comments

```
/* *********************************
 This is a C style comment
*********************************
*/
```

Everything between the /* and the */ is ignored by the compiler. Different programmers have different commenting styles. A word of advice: Programmers are often passionate about how they do business. Rise above the pettiness of arguing commenting issues with fellow programmers. Doing so is a complete waste of mental energy.

However, when using C-style comments keep a few things in mind. They are best used to insert blocks of comments. They can be used to insert one line of comments but C++-style comments are better suited for this purpose as you will see below.

I recommend aligning the /* and */ along the left margin as is shown in the example. You will be less likely to forget the */ and save yourself a lot of wondering why half your program doesn't compile!

Lastly, I also recommend you avoid the urge to make a cute little box out of whatever character you choose to use as a border. For example...

1.6 C-style comments

```
/* ------------------------------------------------
-       This is also a C style comment             -
-                                                  -
------------------------------------------------
*/
```

...only now, if you want to add a line to your comment you have to fiddle around with adding hyphens at the beginning and end of each line.

C++-style Comments

1.7 C++-style comment

```
// This is a C++ style comment
```

As you can see, a C++-style comment begins with two slash characters. They can appear anywhere in your program and tell the compiler to ignore everything that appears to the right up to the end of the line. Another example...

1.8 C++ comment clutter

```
class TestClass{
   public: // public section
      TestClass(); // constructor
      virtual ~TestClass(); //destructor
}; //end of TestClass
```

...shows how to use C++-style comments to really clutter up your code, which leads into a good piece of advice: use them sparingly!

To avoid the need to add comments to your source code in the first place I recommend strongly that you read the next section and take notes.

Write Self-Commenting Code: Give Identifiers Meaningful Names

Self-commenting source code puts the joy back into programming. Self-commenting source code is easier to write, easier to read, easier to maintain, and, if you do happen to make a mistake, your mistake will be easier to find if your source code is self-commenting. How do you self-comment source code?

Essentially, you select names for identifiers that make sense in the context of your program. An identifier is a string of characters used to represent storage locations for variables, constants, functions, types, and other objects within your program.

How you form identifier names is as important as what you name them. Here's some guidance for naming variables, constants, and functions.

Variables

Use lower case letters when declaring variables. Separate each word of a multi-word identifier with an underscore character. Writing variables in lower case will make it easy to spot them in your program. Naming them something that makes sense will remind you of their purpose. The following table gives a few examples, both good and bad, of variable names.

Variable Declaration	Comment
`int a;`	Bad! What the @#%^ does "a" stand for?
`int mother_in_law_count;`	Good! Although you are counting mother-in-laws, at least you know what you are counting.
`Student *s[ 100];`	Bad! How will someone else know that s is an array of pointers to students if they don't see the declaration?
`Student *student_pointers[ 100];`	Good! Now they'll know what's supposed to be in each array element.

Table 1-3: Good vs. Bad Variable Names

Constants

Use upper case letters when defining constants. Separate each word of a multi-word constant with the underscore character. The following table offers a few examples, both good and bad, of constant names.

Constant Declaration	Comment
`const int a = 3;`	Bad! What does a stand for? Is a a variable or a constant?
`const int MAX_ARRAY_SIZE = 100;`	Good!
`#define object_count 25`	Bad! The word count sounds like it might change in the future. Because it is lower case it looks like a variable.
`#define MAX_OBJECT_COUNT 25`	Good! Now it is clear this is a constant and this is the maximum number of objects allowed.

Table 1-4: Good vs. Bad Constant Naming

Functions

Start function names with lower case letters. Join multi-word function names together and capitalize the first letter of each additional word. Functions do things. Verbs denote action. Choose function names that indicate the action the function performs. The following table gives a few examples of function names.

Function Declaration	Comment
`void printScreen();`	Good!
`int getObjectCount();`	Good!
`void print();`	Bad! Print what?
`void setPenPositionUp();`	Good! No mistaking what this function is supposed to do!

Table 1-5: Function Naming

Adopt A Convention And Stick With It

The identifier naming recommendations presented here represent a convention. If you choose to adopt the styles suggested here, fine. If you don't, that's fine too. Whatever naming convention you choose to adopt I recommend you stick with it and be consistent. Don't start naming variables one way and then change the way you name them in the middle of your program. Nothing will confuse you faster than naming inconsistency.

Restrict The Number of Global Variables

Global variables tend to pollute the global name space and lead to the production of tightly coupled code. Tightly coupled code is bad juju, as you will learn below.

Minimize Coupling, Maximize Cohesion

Repeat aloud several times; minimize coupling, maximize cohesion, minimize coupling, maximize cohesion. Good. Practice a few times on your own while I explain why you want to follow this mantra.

Coupling

Coupling refers to the degree to which each module in your source code is affected in any way by making a change to another module. Coupling can be loose, tight, or anywhere in between. You want to keep coupling as loose as possible. How does coupling occur?

One way to couple modules and not even realize you are doing it is through the reckless use of global variables. Modules can also be coupled to other modules, as is the case when one function depends on the services of another function.

It takes considerable knowledge and skill to eliminate all coupling from a group of code modules. For now, be aware that if your code is too tightly coupled, you will break it over there when you make a change here.

Cohesion

Cohesion refers to the degree to which the code in each module contributes to the purpose and function of that module. The rule of thumb is to maximize cohesion. All code belonging to a function should exist to implement that function. Don't do anything surprising or mysterious in a function because it happens to be a convenient place to do it at the time.

Textbooks, Reference Books, and Quick Reference Guides

To be a successful C++ programmer you will need at least three books: A textbook, a language reference book, and a quick reference guide.

What you are reading is a textbook. I put a lot of thought and work into it and as a result I feel it will serve your needs as a textbook very well. However, it is not a language reference book or a quick reference guide to the C++ language. No textbook on the C++ language can be everything to everybody. The C++ standard is over 700 pages long. This book would be huge, and a huge waste of your time, if I tried to include in it everything contained in the standard. Also, when you are in the heat of programming and you just want to quickly see how to declare a class or write a for loop this book will not be the best place to turn.

If you are reading this book you have in your hands a great textbook. In the reference section below I have listed several reference books and quick reference guides I think you will find them very helpful. If it is listed in the reference section I have personally used it and wholeheartedly recommend it.

Summary

The source of a student's difficulty with learning a programming language lies not with the language itself, but with the many other skills that must be mastered almost simultaneously along the way. Students will find it helpful to know the development roles they play and to have a project approach strategy.

The three development roles played by a student are those of analyst, architect, and programmer. As analyst students should strive to understand the project's requirements and what must be done to satisfy those requirements. As architect students are responsible for the design of their project. As programmer, students will implement their project's design in the C++ language.

The project approach strategy helps novice and experienced students systematically formulate solutions to programming projects. The project approach strategy deals with the following areas of concern: requirements, problem domain, language features, and design. By approaching projects in a systematic way, students put themselves in control and can maintain a sense of forward momentum during the execution of their projects. The project approach strategy can be tailored to suit individual needs.

Programming is an art. Formulating solutions to complex projects requires lots of creativity. There are certain steps students can take to stimulate their creative energy. Sketch the project design before sitting at the computer. Reserve quiet space in which to work and, if possible, have a computer dedicated to school and programming projects.

There are five steps to the programming cycle: plan, code, test, integrate, and factor. Use stubbing to test sections of source code without having to code the entire function.

There are two types of complexity: conceptual and physical. Object-oriented programming and design techniques help manage conceptual complexity. Physical complexity is managed with smart project file management techniques and by splitting projects into multiple files.

Use the #ifndef, #define, and #endif preprocessor directives to create header files. Use the #include preprocessor directive to include header files in implementation files.

Self-commenting source code is easy to read and debug. Adopt smart variable, constant, and function naming conventions and stick with them.

Minimize coupling, maximize cohesion!

This is a great textbook! Now, go get a good reference book and quick reference guide.

Skill Building Exercises

1. **Variable Naming Conventions:** Using the suggested naming conventions for variables derive a variable name for each of the concepts listed below:

Number of oranges

Required waivers

Day of week

Month

People in line

Next person

Average age

Student grades

Final grade

Key word

2. **Constant Naming Conventions:** Using the suggested naming convention for constants derive a constant name for each of the concepts listed below:

Maximum student count

Minimum employee pay

Voltage level

Required pressure

Maximum array size

Minimum course load

Carriage return

Line feed

Minimum lines

Home directory

3. **Function Naming Conventions:** Using the suggested naming convention for functions derive a function name for each of the concepts listed below:

Sort employees by pay

List student grades

Clear screen

Run monthly report

Engage clutch

Check coolant temperature

Find file

Display course listings

Display menu

Start simulation

Suggested Projects

1. **Feng Shui:** If you haven't already done so, stake your claim to your own quiet, private space where you will work on your programming projects. If you are planning on using the school's programming lab stop by and familiarize yourself with the surroundings.

2. **Procure and Install IDE:** If you are doing your programming on your own computer make sure you have procured and loaded an integrated development environment that will meet your programming requirements. If in doubt check with your instructor.

3. **Project Approach Strategy Checklist:** Familiarize yourself with the Project Approach Strategy Checklist in Appendix A.

4. **Obtain Reference Books:** Seek your instructor's or a friend's recommendation of any C++ reference books they think will be helpful to you during this course. There are also many good computer book review sites available on the Internet. Also, there are many excellent C++ reference books listed in the reference section of each chapter.

5. **Web Search:** Conduct a web search for C++ and object-oriented programming sites. Bookmark any site you feel might be helpful to you during this class.

Self Test Questions

1. List at least seven skills you must master in your studies of the C++ programming language.

2. What three development roles will you play as a student?

3. What is the purpose of the project approach strategy?

4. List and describe the four areas of concern addressed in the project approach strategy.

5. List and describe the five steps of the programming cycle.

6. What are the two types of complexity?

7. List several benefits to splitting even small projects into multiple files.

8. Discuss the concept of interface vs. implementation. How to you separate the interface of a class from its implementation?

9. What preprocessor directives can be used to allow multiple inclusion of header files?

10. List at least three things that can be contained in header files.

11. List three things that shouldn't be contained in header files.

12. Why do you think it would be helpful to write self-commenting source code?

13. What can you do in your source code to maximize cohesion?

14. What can you do in your source code to minimize coupling?

References

International Standard. ISO/IEC 14882, *Programming Languages — C++*, First edition 1998-09-01. (This is the reference book to the C++ language. You can download it from the American National Standards Institute for a small cost and it is

worth every penny. If you are new to the language it is an extremely daunting document. I also recommend you have a fast Internet connection and an even faster printer!)

Beck, Kent. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Massachusetts, 2000. ISBN 201-61641-6

Lucas, Paul J. *The C++ Programmer's Handbook*, Prentice Hall, Englewood Cliffs, New Jersey, 1992. ISBN 0-13-118233-1 (Great quick reference guide. I put this book in my backpack, take it to class, and show my students on the first day of every C++ class I teach. Age has done nothing to impair the usefulness of this work.)

Ellis, Margaret A., Stroustrup, Bjarne. *The Annotated C++ Reference Manual*, (a.k.a. The ARM), Addison-Wesley, Reading, Massachusetts, 1990. ISBN 0-201-51459-1 (This a great reference book. There is a second edition out now so look for it in the bookstores.)

Notes

