INTRODUCTION TO C++

Chapter 1, Part 1

TOPICS

Overview of C++

Typical C++ Development Environment

Six Phases of C++ Program Development

Introduction to Object Technology

The Software Development Method

OVERVIEW OF C++

History of C++

Developed by Bjarne Stroustrup in 1979 at Bell Labs

Evolved from C

- Developed by Dennis Ritchie at Bell Laboratories
- “C with Classes”

Adds object-oriented programming techniques to C

Evolves the C language for increasingly powerful hardware and more demanding user requirements

Current version is C++14, but C++11 is still in use

C++ Standard Library

C++ programs consist of pieces called classes and functions

Rich collections of pre-built classes and functions in the C++ Standard Library

Three parts to learning C++ programming

- Syntax and use of the language
- Classes and functions in the C++ Standard Library
- Other special-purpose libraries designed for certain problem spaces, like networking

TYPICAL C++ DEVELOPMENT ENVIRONMENT

What You Need to Develop in C++

C++ environment

A way to type up code

C++ Environment Typically Just Called a Compiler

There are many, many compilers out there

Stroustrup's list

https://www.thefreecountry.com/compilers/cpp.shtml

In this class, we'll be using an IDE with the compiler built in

EDITORS

Windows

- Notepad
- Notepad++
- TextPad

macOS

- TextEdit

Linux

- GNU Emacs (for Windows and macOS too)
- vi

Many others…

IDEs

Integrated development environments

- Include editor, compiler, debugger & other helpful tools in one application

Popular IDEs

- Dev-Cpp
- NetBeans
- Eclipse
- Visual Studio
- Xcode
- CodeLite

Six Phases of Program Development

Six Phases of C++ Program Development

C++ programs typically go through six phases

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute

PHASE 1 – EDIT

Type a C++ program (source code) using a text editor

Make any necessary corrections

Save the program.

C++ source code files can use the .cpp, .cxx, .cc, .C, or .h extensions

- Note that C is in uppercase

In this class, all source code will use **.cpp** or **.h**

PHASE 2 – PREPROCESS

Preprocessor program executes automatically before compiler's translation phase begins

Interprets and handles **preprocessor directives**

Preprocessor Directives

Statements at the top of source code files that indicate certain manipulations to be performed on the code before compilation

Can include:

- Other files to be compiled and included
- Text replacements

Appendix E, Preprocessor

PHASE 3 – COMPILE

Compiler **translates** the C++ program into **machine-language code** that represent the tasks to execute

Also referred to as **object code**

PHASE 4 – LINK

Typical C++ program includes references to C++ code defined elsewhere

- C++ Standard Library
- Programmer-defined libraries

Linker **links** the object code of current program with code defined elsewhere to produce an executable program

If the program compiles and links correctly, an executable image is produced

PHASE 5 – LOAD

Program is **placed in memory** by the loader

Additional components from shared libraries that support the program are also loaded

PHASE 6 – EXECUTE

Program **executes**, one instruction at a time

Some modern computer architectures can execute several instructions in parallel

Problems at Execution Time

Programs might not work on the first try.

Each of the preceding phases can fail because of various errors that we'll discuss

If errors occur, return to edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections fixed the problem(s).

INTRODUCTION TO OBJECT TECHNOLOGY

Introduction to Object Technology – Objects

Objects (really, the classes objects come from) **are reusable software components**

- Date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
- Almost any noun can be reasonably represented as a software object in terms of attributes (e.g., name, color and size) and behaviors (e.g., calculating, moving and communicating).

Object-oriented programs are often easier to understand, correct and modify.

The Automobile as an Object – An Analogy

Suppose you want to drive a car and make it go faster by pressing its accelerator pedal.

Before you can drive a car, someone has to design it.

A car typically begins as engineering drawings, similar to the blueprints that describe the design of a house.

Drawings include the design for an accelerator pedal.

Pedal hides from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel hides the mechanisms that turn the car.

The Automobile as an Object – An Analogy, continued

Enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.

Before you can drive a car, it must be built from the engineering drawings that describe it.

A completed car has an actual accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must press the pedal to accelerate the car.

MEMBER FUNCTIONS AND CLASSES

Performing a task in a program requires a grouping of C++ statements called a function

- The function houses the statements that perform tasks

Hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster.

In C++, we create a program unit called a **class** to house the set of functions that perform the class's tasks

A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

INSTANTIATION

Just as someone has to build a car from its engineering drawings before you can actually drive a car, you must build an object from a class before a program can perform the tasks that the class's functions define.

An **object** is then referred to as an **instance** of its class.

REUSE

Just as a car's engineering drawings can be reused many times to build many cars, you can reuse a class many times to build many objects.

Reuse of existing classes when building new classes and programs saves time and effort.

Reuse also helps you build more reliable and effective systems, because existing classes and components should already be tested, debugged and performance tuned.

Reusable classes are crucial to software development

Messages and Function Calls

When you drive a car, pressing its gas pedal sends a message to the car to perform a task—that is, to go faster.

Similarly, you send messages to an object.

Each message is implemented as a **function call** that tells a function of the object to perform its task.

Attributes and Data Members

A car has **attributes**

- Color, number of doors, amount of gas in the tank, current speed, miles driven

The car's attributes are represented as part of its design in its engineering diagrams.

Every car maintains its own attributes.

Each car knows how much gas is in its own gas tank, but not how much is in the tanks of other cars.

Attributes are Specified by Data Members

Attributes are carried along as the object is used in a program.

Specified as part of the object's class.

A bank account object has a balance attribute that represents the amount of money in the account.

Each bank account object knows the balance in the account it represents, but not the balances of the other accounts in the bank.

Attributes are specified by the class's **data members**

ENCAPSULATION

Classes **encapsulate** (i.e., wrap) attributes and functions into **objects**—an object's attributes and functions are intimately related.

Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are hidden within the objects themselves.

Information hiding is crucial to good software engineering.

INHERITANCE

A new class of objects can be created quickly and conveniently by **inheritance**

The new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own.

In our car analogy, an object of class “convertible” is an object of the more general class “automobile,” but with more specific functionality

How Will You Create the Code for Your Programs?

How will you create the code (i.e., the program instructions) for your programs?

Follow a detailed analysis process for determining your project's requirements (i.e., defining what the system is supposed to do)

Develop a design that satisfies them (i.e., deciding how the system should do it).

Carefully review the design (and have your design reviewed by other software professionals) before writing any code.

Object-Oriented Analysis and Design (OOAD)

Analyzing and designing your system from an object-oriented point of view is called an **object-oriented analysis and design** (OOAD) process.

Languages like C++ and Java are object-oriented

Object-oriented programming (OOP) allows you to implement an object-oriented design as a working system.

THE SOFTWARE DEVELOPMENT METHOD

The Software Development Method has Six Steps

Programming is problem-solving!

Steps:

1. Specify
2. Analyze
3. Design
4. Implement
5. Test and Verify
6. Maintain and Update

The Software Development Method – Step 1

Step 1 – Specify the problem requirements.

State the problem clearly.

Zero in on root problem.

THE SOFTWARE DEVELOPMENT METHOD – STEP 2

Step 2 – Analyze the problem.

Identify:

- inputs (what data you have to work with)
- outputs (desired results)
- any additional constraints

Abstraction

- Process of modeling the problem by extracting essential variables and their relationships

THE SOFTWARE DEVELOPMENT METHOD – STEP 3

Step 3. Design the algorithm to solve the problem

Algorithm

- List of steps to solve the problem

Top-Down design (divide & conquer)

- List major steps, then break down major steps into smaller steps

Typical algorithm for a programming problem:

- Get the data
- Perform computations on the data
- Display results

THE SOFTWARE DEVELOPMENT METHOD – STEP 3, CONTINUED

Step 3. Design the algorithm to solve the problem

Stepwise refinement

- Development of a detailed list of steps to solve a particular step in the original algorithm

Desk check

- Step-by-step simulation of the computer execution of an algorithm

THE SOFTWARE DEVELOPMENT METHOD – STEP 4

Step 4. Implement the algorithm

Write the steps as a program, using a programming language

THE SOFTWARE DEVELOPMENT METHOD – STEP 5

Step 5. Test and verify completed program

Run program many times with different sets of test data

Test that program works correctly for every situation provided for in algorithm

THE SOFTWARE DEVELOPMENT METHOD – STEP 6

Step 6. Maintain and update the program

Fix previously undetected errors, modifying it as necessary

Caution: Failure is Part of the Process

- The first solution may be wrong
- But you should be suspicious if your first solution works perfectly…
- Verification of results is vital