Introduction to Programming with C++

Basic Ideas

Objective

- In this chapter we'll discuss:
  - What is meant by Modern C++
  - The elements of a C++ program
  - How to document your program code
  - How your C++ code becomes an executable program
  - How object-oriented programming differs from procedural programming

Modern C++

- Programming using the features of the latest and greatest incarnation of C++.
- The C++ language defined by the C++ 11 standard
- Is being modestly extended and improved by the latest standard, C++ 14.
- In this course we will use C++ as defined by C++14.

Learning C++

- C++ is one of the most widely used and most powerful programming language in the world today.
- It is effective for developing applications across an enormous range of computing devices and environments:
    - Personal Computers
    - Workstations
    - Mainframe Computers
    - Tablets
    - Mobile phones
    - Emended electronics

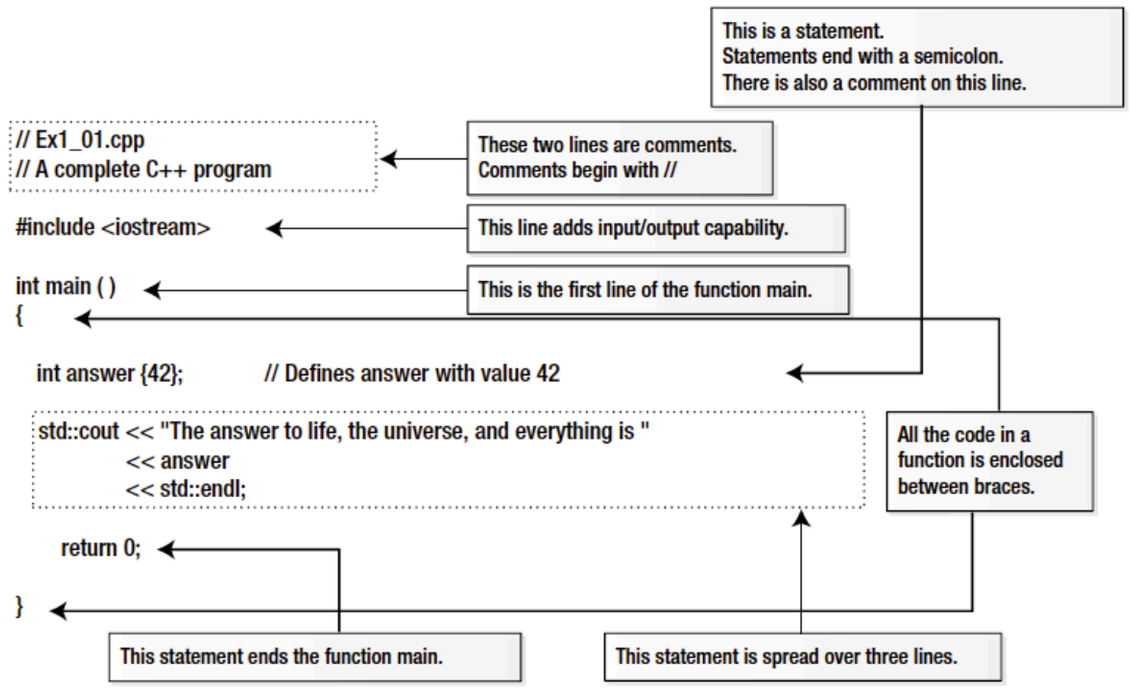
TIOBE Index for March 2019

Mar 2019	Mar 2018	Programming Language	Ratings
1	1	Java	14.880%
2	2	C	13.305%
3	4	Python	8.262%
4	3	C++	8.126%
5	6	Visual Basic .NET	6.429%

Learning C++

- Just about any kind of program can be written in C++ from device drivers to operating systems
- C++ compilers are available widely too.
- C++ comes with a very extensive Standard Library.
- This is a huge collection of routines and definitions that provide functionality that is required by many programs.
    - Numerical calculations
    - String processing
    - Sorting and Searching
    - Organizing and managing data
    - Input and output.

C++ Concepts


This is a statement.
Statements end with a semicolon.
There is also a comment on this line.
// Ex1_01.cpp
// A complete C++ program
These two lines are comments.
Comments begin with //
#include <iostream>
This line adds input/output capability.
int main ( )
This is the first line of the function main.
{
int answer {42};
// Defines answer with value 42
std::cout << "The answer to life, the universe, and everything is "
<< answer
<< std::endl;
All the code in a function is enclosed between braces.
return 0;
}
This statement ends the function main.
This statement is spread over three lines.

Comments and Whitespace

- You add comments that document your program code to make it easier for someone else to understand how it works.
- The compiler ignores everything that follows two successive forward slashes on a line so this kind of comment can follow code on a line.
- There's another form of comment that you can use when you need to spread a comment over several lines. For example:

```
/* This comment is
   over two lines. */
```

- Everything between /* and */ will be ignored by the compiler.

Comments and Whitespace

- You can embellish this sort of comment to make it stand out. For example:

```
/************************
* This comment is *
* over two lines. *
************************/
```

- Whitespace is any sequence of spaces, tabs, newlines, form feed characters, and comments.
- Whitespace is generally ignored by the compiler, except when it is necessary for syntactic reasons to distinguish one element from another.

Preprocessing Directives

- Cause the source code to be modified in some way before it is compiled to executable form.
- The header file contents are inserted in place of the #include directive.
- Header files, which are sometimes referred to just as headers, contain definitions to be used in a source file.
- iostream contains definitions that are needed to perform input from the keyboard and text output to the screen using Standard Library routines.
- In particular, it defines std::cout and std::endl among many other things.

Preprocessing Directives

- We'll be including the contents of one or more standard library header files into every program
- We'll also be creating and using your own header files that contain definitions that you construct later.
- Omitting the include directive from our example would have resulted in compiler time error
- This is because the compiler would not know what std::cout or std::endl are.
- The contents of header files are included into a source file before it is compiled.

Functions

- Every C++ program consists of at least one and usually many more functions.
- A function is a named block of code that carries out a well-defined operation
- Example operations are “read the input data” or “calculate the average value” or “output the results”.
- You execute or call a function in a program using its name.
- All the executable code in a program appears within functions.
- There must be one function with the name main, and execution always starts automatically with this function.

Functions

- The main() function usually calls other functions, which in turn can call other functions, and so on.
- Functions provide several important advantages:
  - A program that is broken down into discrete functions is easier to develop and test.
  - You can reuse a function in several different places in a program, which makes the program smaller.
  - Saves time and effort because of code reuse .
  - Enables team based development for large programs

- The program above consists of just the function main(). The first line of the function is:

```
int main()
```

- This is called the function header, which identifies the function.
- Here, int is a type name that defines the type of value that the main() function returns when it finishes execution - an integer.
- In general, the parentheses following a name in a function definition enclose the specification for information to be passed to the function when you call it.
- There's nothing between the parentheses in this instance but there could be.
- You'll learn how you specify the type of information to be passed to a function when it is executed later in this course.
- The executable code for a function is always enclosed between braces and the opening brace follows the function header.

Statements

- A statement is a basic unit in a C++ program.
- A statement always ends with a semicolon
- It’s the semicolon that marks the end of a statement, not the end of the line.
- A statement defines something, such as a computation, or an action that is to be performed.
- Everything a program does is specified by statements.
- Statements are executed in sequence until there is a statement that causes the sequence to be altered.
- You’ll learn about statements that can change the execution sequence later in this cousre.
-

- There are three statements in main() in the sample presented
- The first defines a variable, which is a named bit of memory for storing data of some kind.
- In this case the variable has the name answer and can store integer values:

```
int answer {42}; // Defines answer with the value 42
```

- The type, int, appears first, preceding the name.
- This specifies the kind of data that can be stored - integers.
- Note the space between int and answer.
- One or more whitespace characters is essential here to separate the type
- name from the variable name;
- Without the space the compiler would see the name intanswer, which it would not
- understand.
- An initial value for answer appears between the braces following the variable name so it starts out storing 42.
- There’s a space between answer and {42} but it’s not essential.
- A brace cannot be part of a name so the compiler can distinguish the name from the initial value specification in any event.
- However, you should use whitespace in a consistent fashion to make your code more readable.
- There’s a somewhat superfluous comment at the end of the first statement explaining what I just described but it does demonstrate that you can add a comments to a statement.
- The whitespace preceding the // is also not mandatory but it is desirable.
-

Statements

- You can enclose several statements between a pair of curly braces, { }, in which case they're referred to as a statement block.
- The body of a function is an example of a block, as you saw in the sample code where the statements in main() function appear between curly braces.
- A statement block is also referred to as a compound statement because in most circumstances it can be considered as a single statement, as you'll see when we look at decision-making capabilities later in this course.
- Wherever you can put a single statement, you can equally well put a block of statements between braces.
- As a consequence, blocks can be placed inside other blocks—this concept is called nesting.
- Blocks can be nested, one within another, to any depth.

Data Input and Output

- Input and output are performed using streams in C++.
- To output something, you write it to an output stream
- To input data you read it from an input stream.
- A stream is an abstract representation of a source of data, or a data sink.
- When your program executes, each stream is tied to a specific device that is
    - the source of data in the case of an input stream
    - the destination for data in the case of an output stream.
- The advantage of having an abstract representation of a source or sink for data is that the programming is then the same regardless of the device the stream represents.
- You can read a disk file in essentially the same way as you read from the keyboard. The standard output and input streams in C++ are called cout and cin respectively and by default they correspond to your computer's screen and keyboard.

- The next statement in main() in the example code outputs text to the screen:

```
std::cout << "The answer to life, the universe, and everything is "
<< answer
<< std::endl;
```

- The statement is spread over three lines, just to show that it's possible.
- The names cout and endl are defined in the iostream header file.
- I'll explain about the std:: prefix a little later in this chapter.
- << is the insertion operator that transfers data to a stream.
- Later on you'll meet the extraction operator, >>, that reads data from a stream.
- Whatever appears to the right of each << is transferred to cout.
- Writing endl to std::cout causes a new line to be written to the stream and the output buffer to be flushed.
- Flushing the output buffer ensures that the output appears immediately.
- The statement will produce the output:
- The answer to life, the universe, and everything is 42
- You can add comments to each line of a statement.
- For example:
- std::cout << "The answer to life, the universe, and everything is " // This statement
- << answer // occupies
- << std::endl; // three lines
- You don't have to align the double slashes but it's common to do so because it looks tidier and makes the code easier to read.

return Statements

- A return statement ends a function and returns control to where the function was called.
- A return statement may or may not return a value.
- Returning 0 to the operating system indicates that the program ended normally.
- You can return non-zero values such as 1, 2, etc. to indicate different abnormal end conditions.
- If execution runs past the last statement in main(), it is equivalent to executing return 0.

- The last statement in main() is a return statement.
- In this case it ends the function and returns control to the operating system.
- This particular return statement returns 0 to the operating system.
- Returning 0 to the operating system indicates that the program ended normally.
- You can return non-zero values such as 1, 2, etc. to indicate different abnormal end conditions.
- The return statement in the example code is optional, so you could omit it.
- This is because if execution runs past the last statement in main(), it is equivalent to executing return 0.

Namespaces

- A large project will involve several programmers working concurrently.
- This potentially creates a problem with names.
- The same name might be used by different programmers for different things, which could at least cause some confusion
- and may cause things to go wrong.
- The Standard Library defines a lot of names, more than you can possibly remember.
- Accidental use of Standard Library names could also cause problems.

Namespaces

- Namespaces are designed to overcome this difficulty.
- A namespace is a sort of family name that prefixes all the names declared within the namespace.
- The names in the standard library are all defined within a namespace that has the name std.
- cout and endl are names from the standard library so the full names are std::cout and std::endl.
- Those two colons together, ::, have a very fancy
- title: the scope resolution operator.
- We'll have more to say about it later.

Namespaces

- The code for a namespace looks like this:

```
namespace ih_space {
   // All names declared in here need to be
   // prefixed with ih_space when they are
   // reference from outside. For example, a
   // min() function defined in here
   // would be referred to outside this
   // namespace as ih_space::min()
}
```

- The main() function must not be defined within a namespace

Names and Keywords

- Lots of things need names in a program and there are precise rules for defining names:
  - A name can be any sequence of upper or lowercase letters A to Z or a to z, the digits 0 to 9 and the underscore character, _.
  - A name must begin with either a letter or an underscore.
  - Names are case sensitive.
- Although it's legal, it's better not to choose names that begin with an underscore;
- they may clash with names from the C++ Standard Library because it defines names in this way extensively.

- The figure presented earlier contains a definition for a variable with the name answer
- it uses the names cout and endl that are defined in the iostream Standard Library header.
-

Names and Keywords

- The C++ standard allows names to be of any length
- but typically a particular compiler will impose some sort of limit.
- However, this is normally sufficiently large that it doesn't represent a serious constraint.
- Most of the time you won't need to use names of more than 12 to 15 characters.

- The figure presented earlier contains a definition for a variable with the name answer
- it uses the names cout and endl that are defined in the iostream Standard Library header.
-

Names and Keywords

- Here are some valid C++ names:

```
toe_count       shoeSize
Box             student
Student         number1
x2              y2
pValue          out_of_range
```

- Uppercase and lowercase are differentiated so `student` is not the same name as `Student` or `STUDENT`.
- You can see a couple of examples of conventions for writing names that consists of two or more words
    - You can capitalize the second and subsequent words
    - or just separate them with underscores.

- The figure presented earlier contains a definition for a variable with the name answer
- it uses the names cout and endl that are defined in the iostream Standard Library header.
-

Names and Keywords

- Keywords are reserved words that have a specific meaning in C++ so you must not use them for other purposes.
- class, double, throw, and catch are examples of keywords.
- You can not use keywords as names for your variables, functions or class names
- Keywords are case sensitive and almost always in lower case
- The full list of keywords in C/C++ is presented in the next slide

alignas	*alignof*	asm	**auto**	bool
break	**case**	catch	**char**	*char16_t*
char32_t	class	**const**	const_cast	*constexpr*
continue	*decltype*	**default**	delete	**do**
double	dynamic_cast	**else**	**enum**	explicit
export	**extern**	false	**float**	**for**
friend	**goto**	**if**	**inline**	**int**
long	mutable	namespace	new	*noexcept*
nullptr	operator	private	protected	public
register	reinterpret_cast	**return**	**short**	**signed**
sizeof	**static**	*static_assert*	static_cast	**struct**
switch	template	this	*thread_local*	throw
true	try	**typedef**	typeid	typename
union	**unsigned**	using	virtual	**void**
volatile	wchar_t	**while**		

Classes and Objects

- A class is a block of code that defines a data type.
- A class has a name that is the name for the type.
- An item of data of a class type is referred to as an object.
- You use the class type name when you create variables that can store objects of your data type.
- Being able to defined you own data types enables you to specify a solution to a problem in terms of the problem.
- If you were writing a program processing information about students for example, you could define a Student type.
- Your Student type could incorporate all the characteristic of a student - such as age, gender, or school record - that was required by the program.

Data Type = Class
Literal = object
int <==> Student
4 <==> s

Templates

- You sometimes need several similar classes or functions in a program where the code only differs in the kind of data that is processed.
- A template is a recipe that you create to be used by the compiler to generate code automatically for a class or function customized for particular type or types.
- The compiler uses a class template to generate one or more of a family of classes.
- It uses a function template to generate functions.
- Each template has a name that you use when you want the compiler to create an instance of it.
- The Standard Library uses templates extensively.

Program Files

- C++ code is stored in two kinds of files Source and Header files.
- Source files contain functions and thus all the executable code in a program.
- The names of source files usually have the extension .cpp, although other extensions such as .cc are also used.
- Header files contain definitions for things such as classes and templates that are used by the executable code in a .cpp file.
- The names of header files usually have the extension .h although other extensions such as .hpp are also used.
- A real-world program may include other kinds of files such as resources that define the appearance of a graphical user interface (GUI).

Standard Libraries

- If you had to create everything from scratch every time you wrote a program, it would be tedious indeed.
- The same functionality is required in many programs
    - For example reading data from the keyboard , or calculating a square root, or sorting data records into a particular sequence.
- C++ comes with a large amount of prewritten code that provides
- facilities such as these
- So you don’t have to write the code yourself.

Standard Libraries

- All this standard code is defined in the Standard Library.
- There is a subset of the standard library that is called the Standard Template Library (STL).
- The STL contains a large number of class templates for creating types for organizing and managing data.
- It also contains many function templates for operations such as sorting and searching collections of data and for numerical processing.

Code Presentation Style

- The way in which you arrange your code can have a significant effect on how easy it is to understand.
- There are two basic aspects to this
  - You can use tabs and/or spaces to indent program statements
  - You can arrange matching braces that define program blocks in a consistent way so that the relationships between the blocks are apparent.
  - You can spread a single statement over two or more lines when that will improve the readability of your program.

Code Presentation Style

- A particular convention for arranging matching braces and indenting statements is a presentation style.
- There are many different presentation styles for code.
- The following slide shows three of many possible options for how a code sample could be arranged

Style 1

```
namespace mine
{
  bool has_factor(int x, int y)
  {
    int f{ hcf(x, y) };
    if (f > 1)
    {
      return true;
    }
    else
    {
      return false;
    }
  }
}
```

Style 2

```
namespace mine{
  bool has_factor(int x, int y)
  {
    int f{ hcf(x, y) };
    if (f > 1) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Style 3

```
namespace mine{
  bool has_factor(int x, int y) {
    int f{ hcf(x, y) };
    if (f > 1){
      return true;
    }
    else{
      return false;
    }
  }
}
```

Creating an Executable

- Creating an executable module from your C++ source code is basically a two-step process.
  - Step I: compiler processes each .cpp file to produce an object file that contains the machine code equivalent of the source file.
  - Step II: linker combines the object files for a program into a file containing the complete executable program.
- Within this process, the linker will integrate any Standard Library functions that you use.
- The image in the next slide shows three source files being compiled to produce three corresponding object files.

Creating an Executable

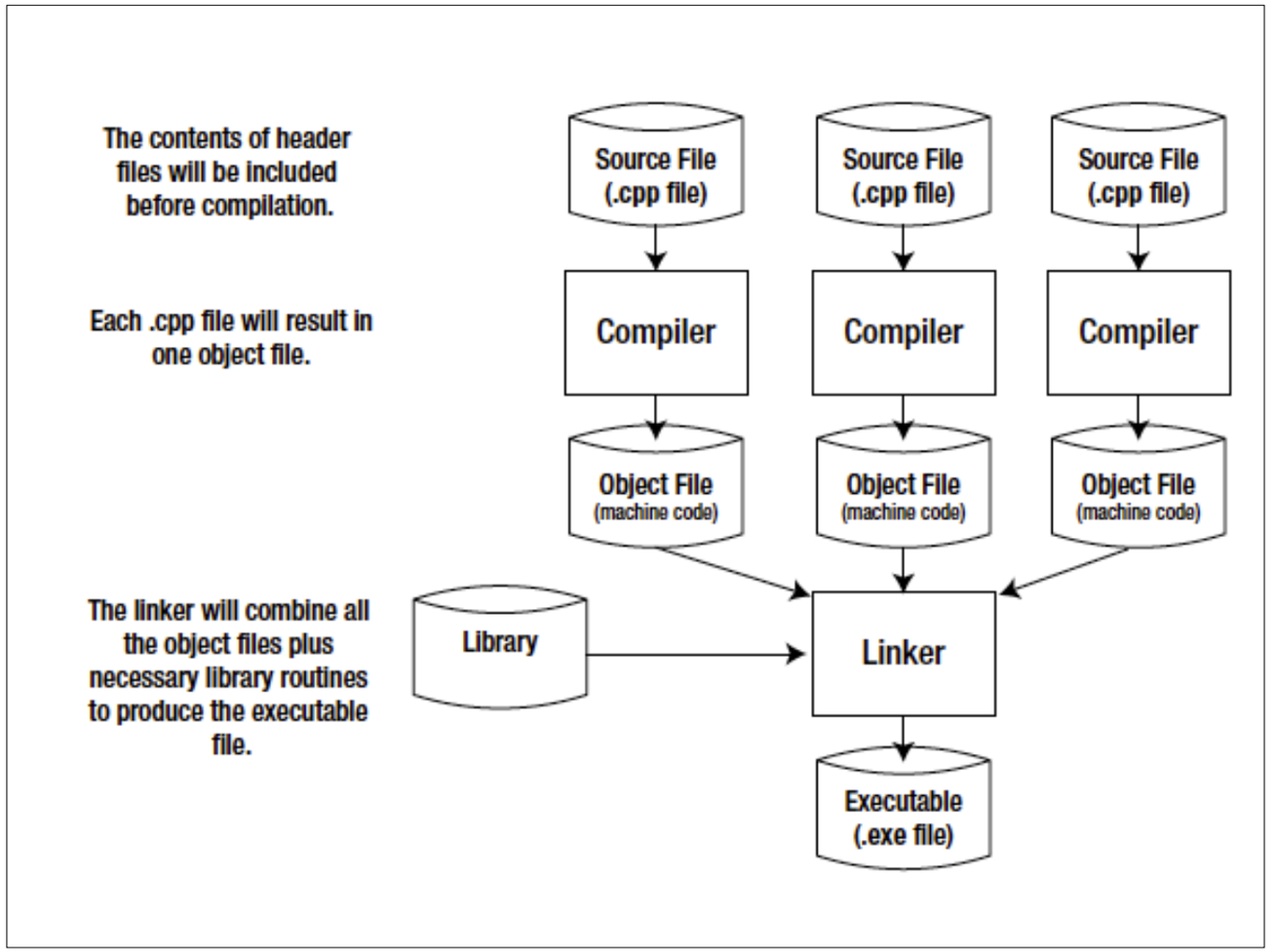
- The filename extension that's used to identify object files varies between different machine environments
- The source files that make up your program may be compiled independently in separate compiler runs or you can compile them all in a single run
- Either way, the compiler treats each source file as a separate entity and produces one object file for each .cpp file.
- The link step then combines the object files for a program, along with any library functions that are necessary, into a single executable file.

Creating an Executable

- In practice, compilation is an iterative process, because you're almost certain to have made typographical and other errors in the code.
- Once you've eliminated these from each source file, you can progress to the link step
- In the link process you may find that yet more errors surface.
- Even when the link step produces an executable module, your program may still contain logical errors
    - It doesn't produce the results you expect.

Creating an Executable

- To fix these, you must go back and modify the source code and try to compile it once more.
- You continue this process until your program works as you think it should.
- As soon as you declare to the world at large that your program works, someone will discover a number of obvious errors that you should have found.
- It hasn't been proven beyond doubt any program larger that a given size will always contain errors.
- It's best not to dwell on this thought when flying.


The contents of header files will be included before compilation.
Each .cpp file will result in one object file.
The linker will combine all the object files plus necessary library routines to produce the executable file.
Source File (.cpp file)
Source File (.cpp file)
Source File (.cpp file)
Compiler
Compiler
Compiler
Object File (machine code)
Object File (machine code)
Object File (machine code)
Library
Linker
Executable (.exe file)

Programming Paradigm

- A programming paradigm is a style, or “way,” of programming.
- Some languages make it easy to write in some paradigms but not others.
- Common paradigms Include
    - Imperative: Programming with an explicit sequence of commands that update state.
    - Declarative: Programming by specifying the result you want, not how to get it.
    - Structured: Programming with clean, goto-free, nested control structures.
    - Procedural: Imperative programming with procedure calls.
    - Functional (Applicative): Programming with function calls that avoid any global state.

Programming Paradigm

  - Object-Oriented: Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces.
  - Event-Driven: Programming with emitters and listeners of asynchronous actions.
  - Flow-Driven: Programming processes communicating with each other over predefined channels.
  - Logic (Rule-based): Programming by specifying a set of facts and rules. An engine infers the answers to questions.
- C is most suited for procedural programming
- C++ added features that made it easy to do OOP to the C language

Procedural Programming

- Focus on the process that your program must implement to solve the problem.
- A rough outline of what you do, once the requirements have been defined precisely, is as follows:
    - You create a clear, high-level definition of the overall process that your program will implement.
    - You segment the overall process into workable units of computation that are, as much as possible, self-contained.
    - These will usually correspond to functions.

Procedural Programming

- You break down the logic and the work that each unit of computation is to do into a detailed sequence of actions.
- This is likely to be down to a level corresponding to programming language statements.
- You code the functions in terms of processing basic types of data:
    - numerical data
    - single characters
    - character strings.

Object Oriented Approach

- From the problem specification, determine what types of objects the problem is concerned with.
- For example, if your program deals with baseball players, you're likely to identify BaseballPlayer as one of the types of data your program will work with.
- If your program is an accounting package, you may well want to define objects of type Account and type Transaction.
- You also identify the set of operations that the program will need to carry out on each type of object.
- This will result in a set of application-specific data types that you will use in writing your program.

Object Oriented Approach

- You produce a detailed design for each of the new data types that your problem requires, including the operations that can be carried out with each object type.
- You express the logic of the program in terms of the new data types you've defined and the kinds of operations they allow.
- The program code for an object-oriented solution to a problem will be completely unlike that for a procedural solution and almost certainly easier to understand.
- It will also be a lot easier to maintain.

Object Oriented Approach

- You produce a detailed design for each of the new data types that your problem requires, including the operations that can be carried out with each object type.
- You express the logic of the program in terms of the new data types you've defined and the kinds of operations they allow.
- The program code for an object-oriented solution to a problem will be completely unlike that for a procedural solution and almost certainly easier to understand.
- It will also be a lot easier to maintain.
- The amount of design time required for an object-oriented solution tends to be greater than for a procedural solution.
- However, the coding and testing phase of an object-oriented program tends to be shorter and less troublesome
- So the overall development time is likely to be roughly the same in either case.

Summary

- Some of the basics that this chapter covered are as follows:
- A C++ program consists of one or more functions, one of which is called main().
- Execution always starts with main().
- The executable part of a function is made up of statements contained between braces.
- A pair of curly braces is used to enclose a statement block.
- A statement is terminated by a semicolon.
- Keywords are reserved words that have specific meanings in C++.
- No entity in your program can have a name that coincides with a keyword.

Summary

- A C++ program will be contained in one or more files.
- Source files contain the executable code and header files contains definitions used by the executable code.
- The source files that contain the code defining functions typically have the extension .cpp.
- Header files that contain definitions that are used by a source file typically have the extension .h.
- Preprocessor directives specify operations to be performed on the code in a file.
- All preprocessor directives execute before the code in a file is compiled.

Summary

- The contents of a header file is added to a source file by a #include preprocessor directive.
- The Standard Library provides an extensive range of capabilities that supports and extends the C++ language.
- Access to Standard Library functions and definitions is enabled through including Standard
- Library header files into a source file.
- Input and output is performed using streams
- It involves the use of the insertion and extraction operators, << and >>.
- std::cin is a standard input stream that corresponds to the keyboard.

Summary

- std::cout is a standard output stream for writing text to the screen.
- Both are defined in the iostream Standard Library header.
- Object-oriented programming involves defining new data types that are specific to your problem.
- Once you've defined the data types that you need, a program can be written in terms of the new data types.

Exercise

1. Create, compile, link, and execute a program that will display the text "Hello World" on your screen.
2. Create and execute a program that outputs your name on one line and your ID on the next line.
3. The following program produces several compiler errors. Find these errors and correct them so the program can compile cleanly and run.

```
include <iostream>

Int main()
{
  std:cout << "Hello World" << std:endl
)
```

Reading Assignment

1. Read about data representation in computers, particularly
   - Representing Numbers
     - Binary Numbers
     - Hexadecimal Numbers
     - Negative Binary Numbers
     - Octal Values
     - Big-Endian and Little-Endian Systems
     - Floating-Point Numbers
   - Representing Characters
     - ASCII Codes
     - UCS and Unicode

Reading Assignment

2. Read about the c preprocessor, in particular

#include	#error
#define	__FILE__
#undef	__LINE__
#if	__DATE__
#ifdef	__TIME__
#ifndef	__TIMESTAMP__