SWE20004
Technical Software Development

Lecture 1
Introduction to Fundamental Concepts

Slides credit : Dr Recep Ulusoy
Modified by: Dr. Prince Kurumthodathu Surendran

Outline

- Administration of the unit
- What we are going to cover in this unit
- Revision of programming concepts
- Software develoment
- Algorithm development and representation
- Overview of programming languages
- Practical programming and associated tools

Staff

- Prince Kurumthodathu Surendran
  - Lecturer
  - pkurumthodathusurend@swin.edu.au
  - TD192 (Ph: 9214 5421)
- Tutors
  - Gavin Chan (gchan@swin.edu.au) – Wed 10.30 am in BA601
  - Rida Fatima (rfatima@swin.edu.au) – Wed 12.30 pm in BA601
  - Zyeda Zehra (szehra@swin.edu.au) – Wed 2.30 pm (BA405) & Thu 4.30 pm (EN402)
  - Srikanth Thudumu (sthudumu@swin.edu.au) – Wed 4.30 pm
  - Kai Renshaw (krrenshaw@swin.edu.au) - Thu 12.30 pm in EN402
  - Anika Kanwal (akanwal@swin.edu.au) – Thu 2.30 pm in EN402
  - Michael George (mgeorge@swin.edu.au) – Fri 12.30 pm in ATC009

- Help Desk hours – See Canvas

The Unit Outline

- On Canvas
- Contains details of assessment dates
- Provisional outline of weekly tasks
- References to textbook and other resources
- Other important stuff…

Assessment

- Assignments (3) - 20%
- Lab tests (2) - 20%
- Exam - 60%

Total: 100%

Must get minimum 40% in the final exam (24 out of 60 marks) as well as minimum 50% overall in order to pass the subject…

Textbook

- Bronson G. J. (2013). C++ for Engineers and Scientists (4th ed), Cengage.
- Other C++ books OK, but the readings, homework and most of the other things come from this book.

The Slides

- Available each week on Canvas as pdfs
- Each Lecture:
  - Print them, bring them with you, write on them
- Each Lab:
  - Bring them, refer to them, use them too
- When you study/cram for test, exam:
  - Read them
  - Summarise them
  - Understand the concepts covered in them

Assumptions

We will assume that:

- You are familiar with basic programming theory, you know what:
  - sequence is
  - selection is
  - iteration/repetition is
- You can do basic arithmetic/algebra
- You understand what operators, operands and expressions are
- You understand basic *boolean* logic

We will use C++ language

- C++ uses C syntax, so familiarity with some C will be an advantage, but we will cover the syntax
- We will use C++ *I/O streams* instead of *stdio.h* of C
- We will use the '*string*' class instead of C-strings.
- We will use '*references*' instead of '*pointers*' although we may still introduce them towards the end of the unit to see the difference

C++ is just as fast as C

- C++ uses the same compiler as C, GNU C++ (gcc or g++)
- Uses different libraries (if you want to)
- Supports all of C
- C++ is sort-of portable
- If you use standard C++ libraries, you get portable code.
  - Just re-compile on the target platform.
  - Good for Windows, Unix, Linux, Mac OS X

This is not a complete OOP subject

- No comprehensive coverage of object oriented concepts and constructs
- Just the basic introduction to classes and objects with their constituent parts towards the end of the unit
- Perhaps some inheritance with base (super) and derived (sub) classes if time permits
- Probably no friend functions, operator overloading, virtual classes, interfaces, or detailed coverage of visibility specifiers with public, private, protected functions unless there is time left for them

Software Development

- **Computer program:** Self-contained set of instructions used to operate a computer to produce a specific result
  - Also called **software**
  - Solution developed to solve a particular problem, written in a form that can be executed on a computer

Software Development (continued)

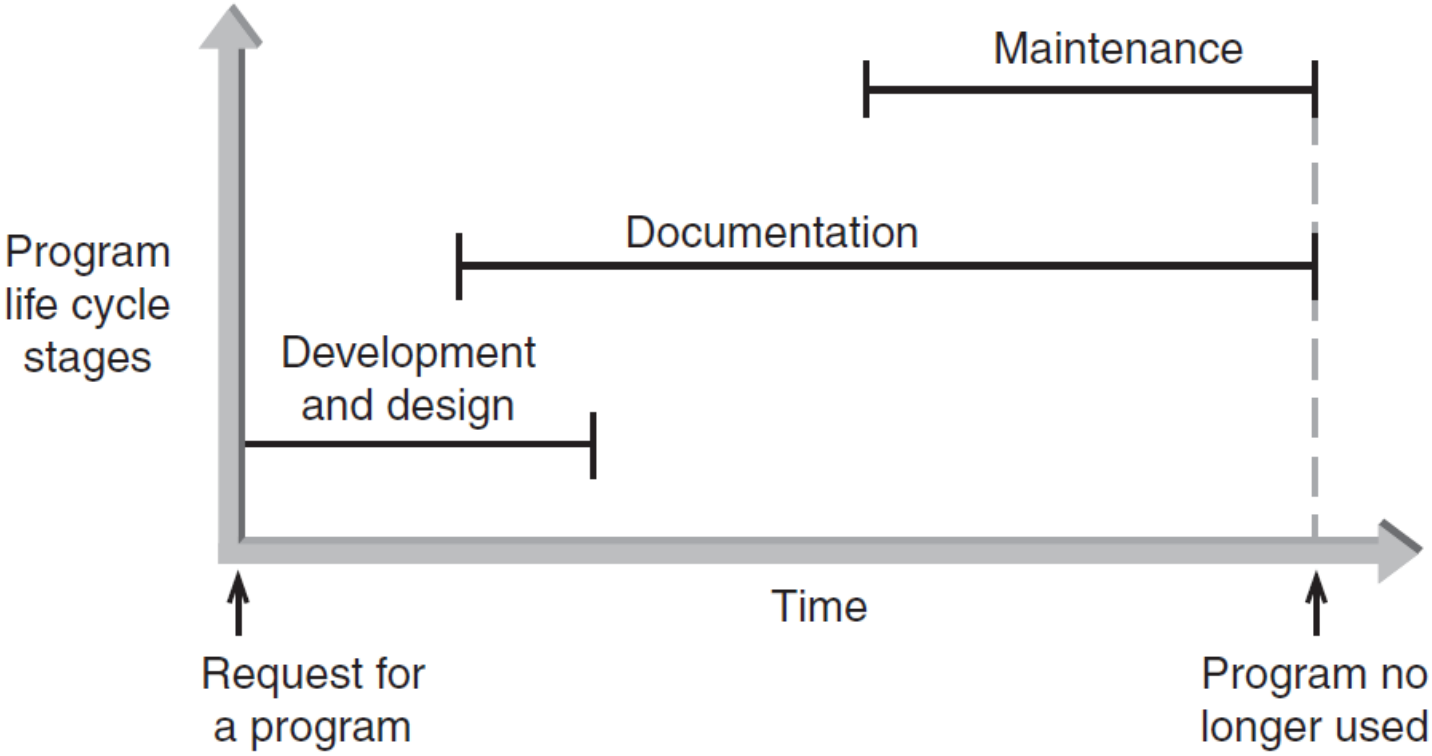


Figure 1.2 The three phases of program development

Phase I: Development and Design

- **Program requirement**: Request for a program or a statement of a problem
- After a program requirement is received, Phase I begins:
- Phase I consists of four steps:
  - Analysis
  - Design
  - Coding
  - Testing

Phase I: Development and Design (continued)

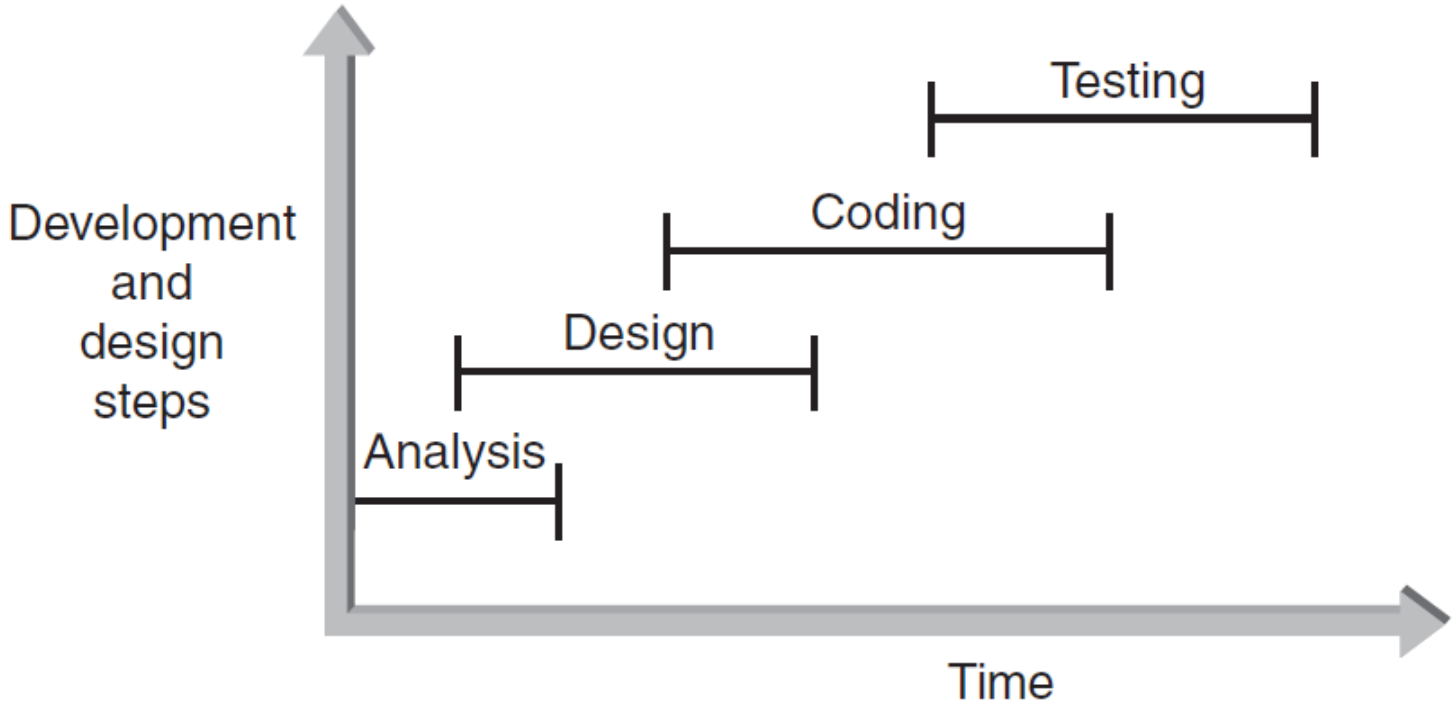


Figure 1.3 The development and design steps

Phase I: Development and Design (continued)

- Step 1: Analyze the Problem
  - Determine and understand the output items the program must produce
  - Determine the input items
  - Both items referred to as the problem's input/output (I/O)

Phase I: Development and Design (continued)

- Step 2: Develop a Solution
  - Select the exact set of steps, called an "algorithm," to solve the problem
  - Refine the algorithm
    - Start with initial solution in the analysis step until you have an acceptable and complete solution
  - Check solution

Phase I: Development and Design (continued)

- Step 3: Code the Solution
  - Consists of actually writing a C++ program that corresponds to the solution developed in Step 2
  - Program should contain well-defined patterns or structures of the following types:
    - Sequence
    - Selection
    - Iteration
    - Invocation

Phase I: Development and Design (continued)

- Step 3: Code the Solution (continued)
  - **Sequence:** Defines the order in which instructions are executed
  - **Selection:** Allows a choice between different operations, based on some condition
    - Conditional if/else statements
  - **Iteration:** Allows the same operation to be repeated based on some condition
    - Also called looping or repetition
  - **Invocation:** Involves invoking a set of statements when needed
    - Calling/invoking functions

Phase I: Development and Design (continued)

- Step 4: Test and Correct the Program
  - **Testing**: Method to verify correctness and that requirements are met
  - **Bug**: A program error
  - **Debugging**: The process of locating an error, and correcting and verifying the correction
  - Testing may reveal errors, but does not guarantee the absence of errors

Phase II: Documentation

- Five main documents for every problem solution:
  - Program description
  - Algorithm development and changes
  - Well-commented program listing
  - Sample test runs
  - User manual

Phase III: Maintenance

- **Maintenance** includes:
  - Ongoing correction of newly discovered bugs
  - Revisions to meet changing user needs
  - Addition of new features
- Usually the longest phase
- May be the primary source of revenue
- Good documentation vital for effective maintenance
- Backup copies to cover for loss or damage

Algorithm development and representation

- **Algorithm:** Step-by-step sequence of instructions
  - Must terminate
  - Describes how the data is to be processed to produce the desired output
- **Pseudocode:** English-like phrases used to describe steps in an algorithm
- **Formula:** Mathematical equations
- **Flowchart:** Diagrams with symbols

Algorithms (continued)

- Problem: Calculate the sum of all whole numbers from 1 to 100

Method 1 - Columns: Arrange the numbers from 1 to 100 in a column and add them.

```
  1
  2
  3
  4
  .
  .
  .
 98
 99
+100
-----
5050
```

Figure 1.6 Summing the numbers 1 to 100

Algorithms (continued)

Method 2 - Groups: Arrange the numbers in groups that sum to 101 and multiply the number of groups by 101.

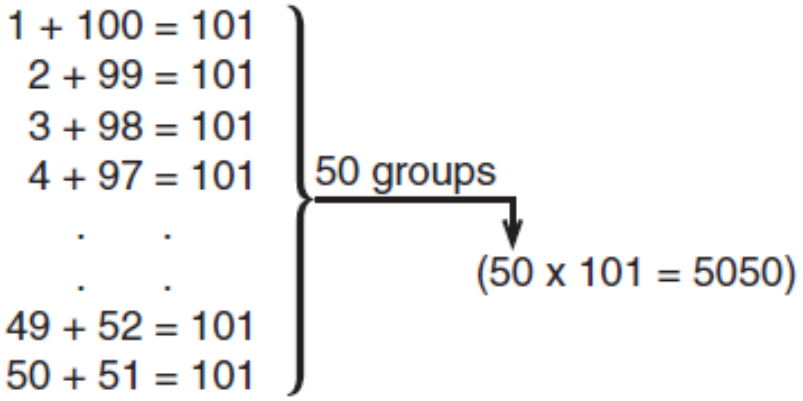


Figure 1.6 Summing the numbers 1 to 100 (continued)

Algorithms (continued)

Method 3 - Formula: Use the formula.

$$\text{sum} = \frac{n(a + b)}{2}$$

where

n = number of terms to be added (100)
a = first number to be added (1)
b = last number to be added (100)

$$\text{sum} = \frac{100(1 + 100)}{2} = 5050$$

Figure 1.6 Summing the numbers 1 to 100 (continued)

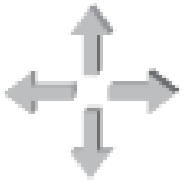
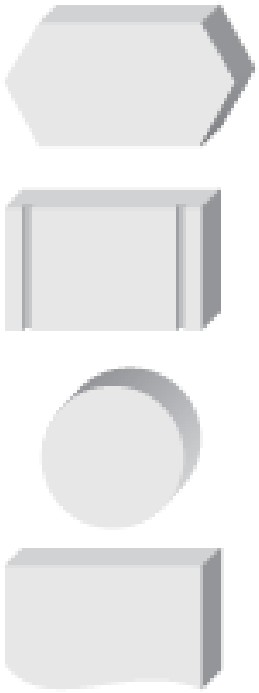
Symbol	Name	Description
	Terminal	Indicates the beginning or end of a program
	Input/output	Indicates an input or output operation
	Process	Indicates computation or data manipulation
	Flow lines	Used to connect the other flowchart symbols and indicate the logic flow
	Decision	Indicates a program branch point

Figure 1.7 Flowchart symbols



Loop	Indicates the initial, limit, and increment values of a loop
Predefined process	Indicates a predefined process, as in calling a function
Connector	Indicates an entry to, or exit from, another part of the flowchart or a connection point
Report	Indicates a written output report

Figure 1.7 Flowchart symbols (continued)

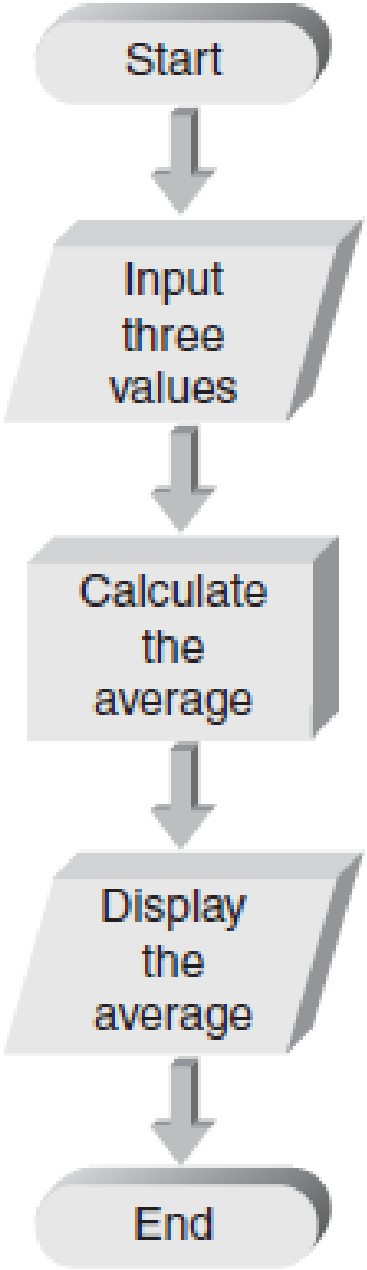


Figure 1.8 Flowchart for calculating the average of three numbers

Programming languages and styles

- **Programming:** Process of writing a program, or software
- **Programming language:**
  - Set of symbols, words and rules used to construct a program
  - Comes in a variety of forms and types

Machine Language

- **Machine language programs**: only programs that can actually be directly used to operate a computer
  - Only language that a particular machine architecture understands
  - Also referred to as executable programs (executables)
  - Consists of a sequence of instructions composed of binary numbers (sequence of 0s and 1s)
  - Contains two parts: an instruction (aka opcode) and an address

Assembly Language

- **Assembly language programs**: Substitute word-like symbols, such as ADD, SUB, and MUL, for binary opcodes
  - Use decimal numbers and labels for memory addresses
    - Example: `ADD 1, 2`
- **Assemblers:** Translate programs into machine language

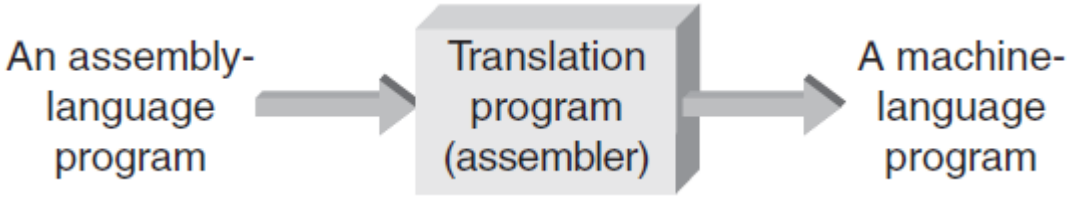


Figure 1.10 Assembly-language programs must be translated

Low- and High-Level Languages

- **Low-level languages:** Languages that use instructions tied directly to one type of computer
  - Examples: machine language, assembly language
- **High-level languages:** Instructions resemble written natural/human languages (mostly English)
  - Can be run on a variety of computer types
  - Examples: C, C++, Java, Python, LISP, Prolog

Low- and High-Level Languages (continued)

- **Source code:** The programs written in a high- or low-level language
  - Before execution, source code must be translated to machine instructions in one of two ways, using:
    - **Interpreter:** Each statement is translated individually and executed immediately after translation
      - Python, LISP and Prolog are examples of interpreted languages
    - **Compiler:** All statements are translated and stored as an executable program, or object program; execution occurs later
      - C, C++ are examples of compiled language
  - Some languages (such as Java) use both: source is first translated to an intermediate code (eg. byte code) using a compiler and then executed using an interpreter (eg. JVM – Java Virtual Machine)

Low- and High-Level Languages (continued)

- Large C++ programs may be stored in two or more separate program files (modules) due to:
  - Use of previously written code
  - Use of code provided by the compiler
  - Modular design of the program (for reusability of components)
- **Linker:** Combines all of the compiled code required for the program

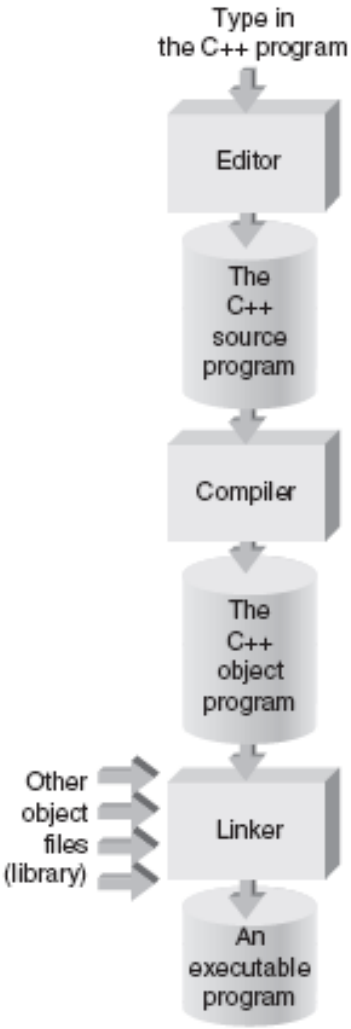
Procedural and Object Orientations

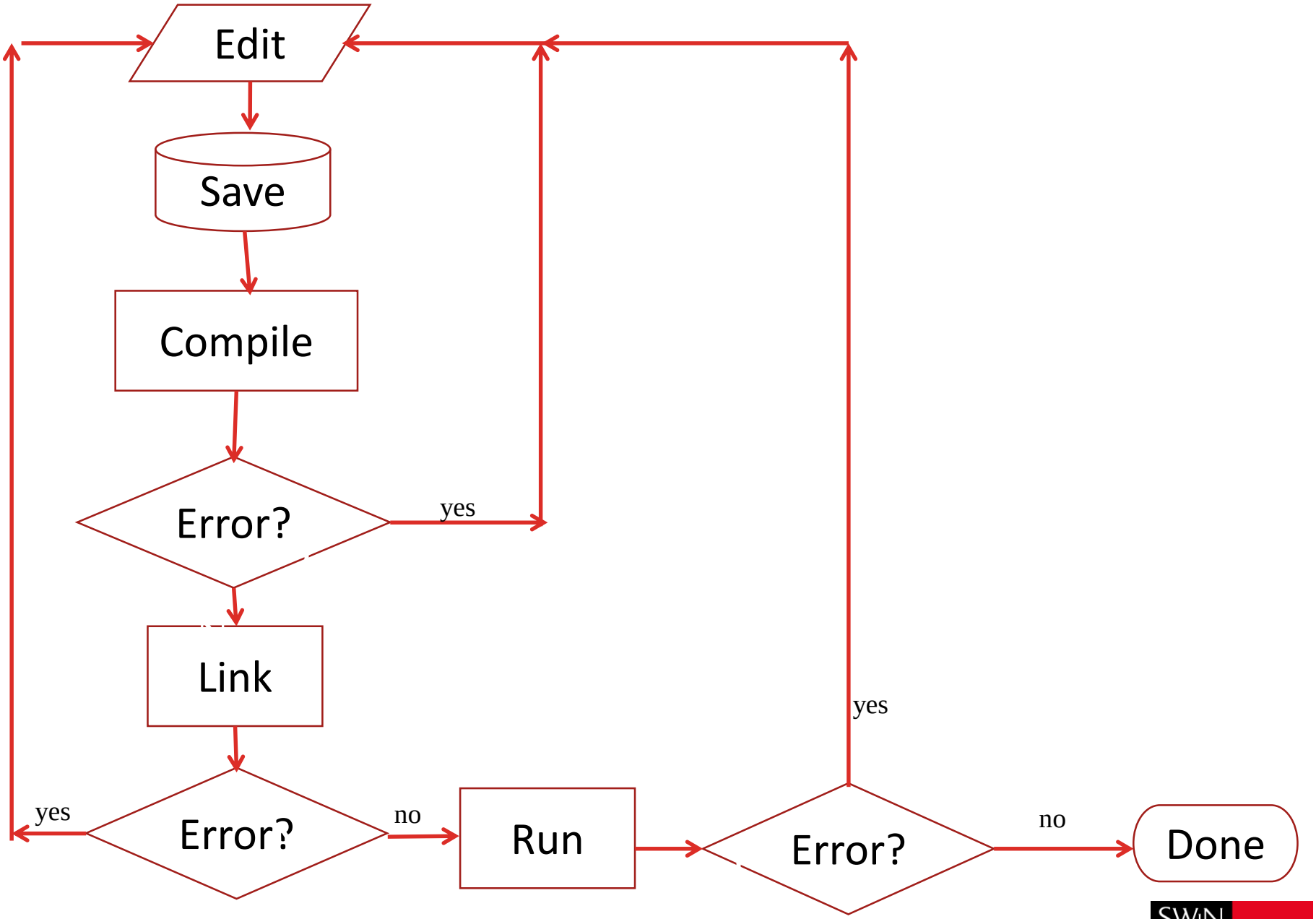
- Programs can also be classified by their orientation:
  - **Procedural:** Instructions are used to create self-contained units called procedures (or functions) that process data
  - **Object-oriented:** Program is built of reusable objects, containing both code and data to be processed
    - Object-oriented languages support reusing existing code more easily
- C++ contains features of both so in this sense it is a hybrid language

Practical coding

Steps involved in creating and running programs:

1. Write/Edit the source code using a text editor
2. Save the file
3. Compile the code
   - compiler checks for errors
   - if no error, run the linker
4. Run/execute the program
5. If anything goes wrong go back to step 1




Edit
Save
Compile
Error?
yes
Link
Error?
yes
no
Run
Error?
yes
no
Done

Running C++

C++ can be produced and run using:

- a text editor and a stand alone compiler and linker (gcc or g++)
  - On Linux, Windows, Mac
- an IDE (Integrated Development Environment)
  - Eclipse, NetBeans, Visual Studio, Visual C++ Express, Quincy, Code::Blocks, DevC++ (Windows)
  - Xcode, Eclipse, NetBeans, Code::Blocks (Mac)
  - Eclipse, NetBeans, Code::Blocks (Linux)

C++ requires no IDE

- Like C, you can develop C++ with a simple text editor and a compiler **[this is the recommended method in this unit so use it whenever possible]**
- No need for fancy products (with big learning curves)
  - Visual Studio, Code::Blocks, DevC++, MinGWStudio, Eclipse, X-Code
- Just use a simple text editor and learn to compile and execute programs from command windows on a particular platform

Popular text editors

- Windows
  - Crimson Editor
  - Notepad++
  - TextMate
  - Sublime
- Mac
  - TextMate
  - TextWrangler
  - Sublime
  - Vi/Vim
- Linux
  - Vi/Vim
  - Emacs
  - gEdit
  - gVim
  - Nano

C++ compilers

- MinGW (g++, gcc) for Windows
  - http://sourceforge.net/projects/mingw-w64/

- After installation, remember to update the path variable to include the bin folder where g++.exe is located (eg. C:\Program Files\mingw32\bin)

- Search for gcc or g++ for Mac or Linux if not already installed

Windows command line

- Create/edit your file using editor (eg. Crimson or **Notepad++**):
- Open a Terminal window (you can do this from within the editor in some cases)
- Compile:

  g++ example.cpp -o myProgram

- Run:

  myProgram

Running C++ in Visual Studio or Visual C++ Express

- If you prefer to use Visual Studio:
  - There are many online tutorials to help you with this. See for example: http://www.cplusplus.com/doc/tutorial/introduction/visualstudio/
  - Visual C++ and Visual Studio Tutorials in Week-01 lab materials on Blackboard
  - Read all the instructions carefully

Linux / Mac command line

- Create/edit your file using editor (eg. vi):

  `vi example.cpp`

  vi commands:
  - i – insert mode
  - esc – command mode
  - [esc] :wq – save and exit
- Open a Terminal window
- Compile:

  g++ example.cpp -o myProgram
- Run:

  ./myProgram

`vi` commands can be looked up on the web. Linux has GUI-based editors too.

`Or use`
`gcc instead of g++`

Need assistance?

- Textbooks
- Online web resources such as www.cplusplus.com
- Tutors
- Canvas discussion board
- Programming help desk (ATC620)