Program Development Cycle

- Many programmers follow a sequence of Steps to create their programs.

1. Analyze – Define the Problem
   - Make sure that you understand what the program should do. What should the user be able to enter? How? How does the program come up with an answer? What does the program output? How?
     - User – a person who uses a computer program.
     - End User – the user that the program was made for.
2. Design – Plan a Solution for the Problem
   - Develop a PRECISE sequence of steps to solve the problem
     - An algorithm is a precise sequence of steps to solve a problem.

Program Development Cycle – Design (Contd.)

- Typically a program follows three general steps
  1. Input
  2. Processing (Formulas)
  3. Output

Develop an Algorithm

- Imagine you want a program that tells a user how many stamps they need in order to mail a certain number of pages.
- You need one stamp for every 5 pages
  - 6 pages = 2 stamps
  - 12 pages = 3 stamps
  - …
- Write an algorithm (the steps needed) to solve this problem

Algorithm

- Algorithm is a ***finite*** set of ***unambiguous instructions*** which when executed, performs a task correctly.
- Three characteristic features for description of algorithm are:
- The number of steps required to perform the task should be finite.
- Each of the instruction in the algorithm should be unambiguous in nature (Outcome should be definite and predictable)
- Algorithm should solve the problem correctly.

Example (Use of imperative constructs)

Problem- Develop an algorithm to find average of three numbers taken as input from the user

Algorithm AVERAGE

STEP 0 START
STEP 1 INPUT first number into variable A
STEP 2 INPUT second number into variable B
STEP 3 INPUT second number into variable C
STEP 4 COMPUTE SUM= A+B+C
STEP 5 COMPUTE AVG=SUM/3
STEP 6 DISPLAY AVG
STEP 7 END

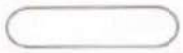
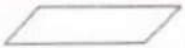
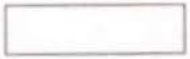
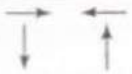
Programming Tools: Pseudocode

- ***Pseudocode***: an abbreviated plain English version of actual computer code. Kind of a mix between English and code. THERE IS NO OFFICIAL SYNTAX TO PSEUDOCODE. Pseudocode may be considered to be an intermediate code between an algorithm and program code. Specifications used in pseudocode are syntactically not stringent as in a program code.
- ***Example:*** Pseudocode to find sum of first N natural numbers where N would be input by the user

```
Pseudocode SUMN
{
print “Enter a value for the number N: ”
scan N
I = 1
SUM = 0
While I <+ N
{
SUM = SUM + I
I = I + 1;
}
print “The sum is: ”, SUM
}
```

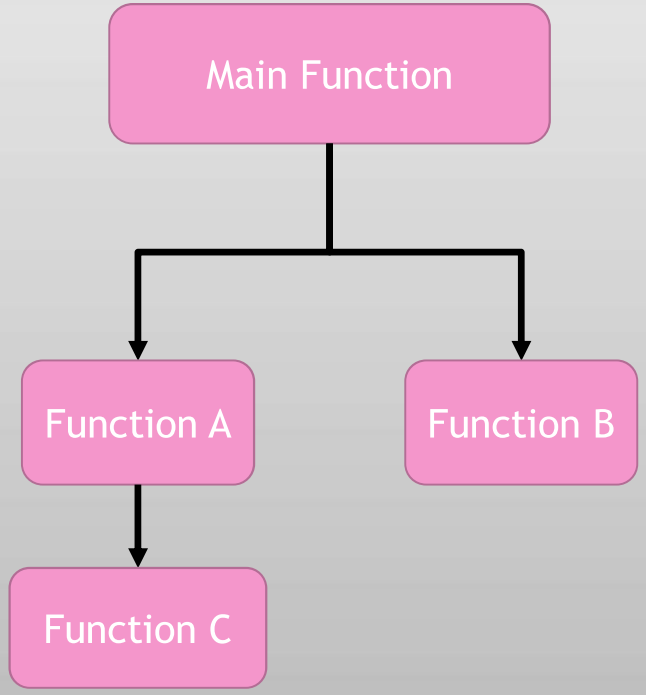
Programming Tools: Flowchart

- ***Flowcharts*** – A chart that consists of symbols connected by arrows. Within each symbol is a phrase presenting the activity at that step. The shape of the symbol indicates the type of operation that is to occur.
- Helps to graphically visualize the flow of control within the sequence of statements.

Name	*Description*	*Symbol*
Terminal symbol	This symbol is used to mark the beginning and end of a flowchart.	
Input/output box	This box is used to represent input operations or output operations.	
Process box	This box is used to represent processing operations. More specifically, this symbol is used to represent the imperative logic construct.	
Decision box	This diamond-shaped symbol is used to represent conditional statement constructs. Each such box has two exit points which correspond to the evaluation of the condition to true and false.	
Connector (intra-page)	This symbol is used to transfer the flow of control from one point to another within a page. A connector is usually labeled to match with its counterpart.	
Off-page connector (inter-page)	This symbol is used to transfer the flow of control from one point on a page to another point on a different page. This notation is useful when the flowchart spans more than one page.	
Arrowed lines	These are used to show the sequence of flow from one box to another.	

Programming Tools: Hierarchy chart

- ***Hierarchy charts*** – A chart also known as a structure chart or top-down chart, is a graphical representation of the structure and organization of a program's modules or functions that shows the overall program structure. These charts describe what each part, or module, does and how they are related. These modules intentionally omit details of how they work.



Program Development Cycle

3. Write the Code – Implement a solution
   - The instructions in a programming language collectively called code.
   - Your code should be a translation of your algorithm developed into the programming language.
     - In this class we use Java, but there are many other programming languages: C, C++, C#, Ruby, Python, Visual Basic, etc.
   - This is the major focus of this course, but note that you need to be able to think algorithmically in order to do this.
     - Meaning, you need to be able to logically solve the problem in order to write a program for it.

Program Development Cycle

4. Testing and Debugging – Locate and remove any errors in the program
   - Testing is the process of finding errors in a program
   - Debugging is the process of removing errors in a program.
     - An error in a program is called a bug.
   - We will talk more specifically about the kinds of errors that can occur in a program once we start programming.
5. Complete All Documentation – Organize the material that describes the program.
   - Documentation is any material whose purpose is to allow another person or programmer to use or understand the program
   - Two kinds of documentation:
     1. External Documentation – Material outside of the code files that describe the program.
     2. Internal Documentation – Lines inside of a code file that do nothing except describe details of the program. In Java, these are called comments.

Programs

- Programs are written in programming languages
  - PL = programming language
  - Pieces of the same program can be written in different PLs
    - Languages closer to the machine can be more efficient
    - As long as they agree on how to communicate
- A PL is
  - A special purpose and limited language
  - A set of rules and symbols used to construct a computer program
  - A language used to interact with the computer

Computer Languages

- Machine Language
  - Uses binary code
  - Machine-dependent
  - Not portable
- Assembly Language
  - Uses mnemonics
  - Machine-dependent
  - Not usually portable
  - LOAD, ADD, MUL, MOV
- High-Level Language (HLL)
  - Uses English-like language
  - Machine independent
  - Portable (but must be compiled for different platforms)
  - Examples: Pascal, C, C++, Java, Fortran, Python,. . .

Machine Language

- The representation of a computer program which is actually read and understood by the computer.
  - A program in machine code consists of a sequence of machine instructions.
- Instructions:
  - Machine instructions are in binary code
  - Instructions specify operations and memory cells involved in the operation

Example:

Operation	Address
0010	0000 0000 0100
0100	0000 0000 0101
0011	0000 0000 0110

Assembly Language

- A symbolic representation of the machine language of a specific processor.
- Is converted to machine code by an assembler.
- Usually, each line of assembly code produces one machine instruction (One-to-one correspondence).
- Programming in assembly language is slow and error-prone but is more efficient in terms of hardware performance.
- Mnemonic representation of the instructions and data
- **Example:**

```
Load    Price
AddTax
Store   Cost
```

High-level language

- A programming language which use statements consisting of English-like keywords such as "FOR", "PRINT" or “IF“, ... etc.
- Each statement corresponds to several machine language instructions (one-to-many correspondence).
- Much easier to program than in assembly language.
- Data are referenced using descriptive names
- Operations can be described using familiar symbols
- Example:

  Cost := Price + Tax

Syntax & Semantics

- Syntax:
  - The structure of strings in some language. A language's syntax is described by a grammar.
  - Examples:
    - Binary number

      <binary_number> = <bit> | <bit> <binary_number>

      <bit> = 0 | 1
    - Identifier

      <identifier> = <letter> {<letter> | <digit> }

      <letter> = a | b | . . . | z

      <digit = 0 | 1 | . . . | 9
- Semantics:
  - The meaning of the language

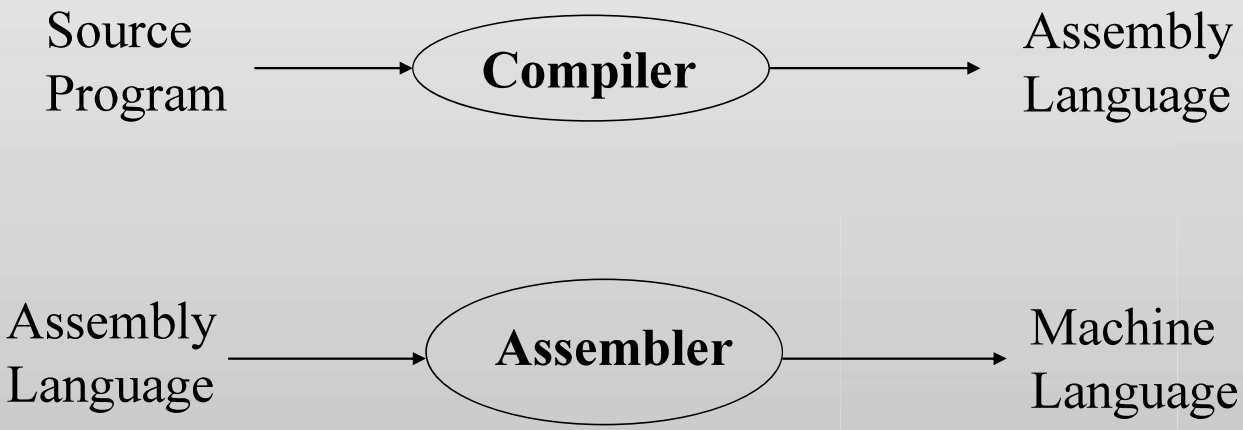
Syntax & Grammars

- Syntax descriptions for a PL are themselves written in a formal language.
  - E.g. Backus-Naur Form (BNF)
- The formal language is not a PL but it can be implemented by a compiler to enforce grammar restrictions.
- Some PLs look more like grammar descriptions than like instructions.

Compilers & Programs

- **Compiler**
  - A program that converts another program from some source language (or high-level programming language / HLL) to machine language (object code).
  - Some compilers output assembly language which is then converted to machine language by a separate assembler.
  - Is distinguished from an assembler by the fact that each input statement, in general, correspond to more than one machine instruction.
  - Overview of Compiler working:
    - **Preprocessing**
    - **Parsing**
    - **Semantic Analysis**
    - **Intermediate Code Generation**
    - **Code Optimization**

Compilation into Assembly L

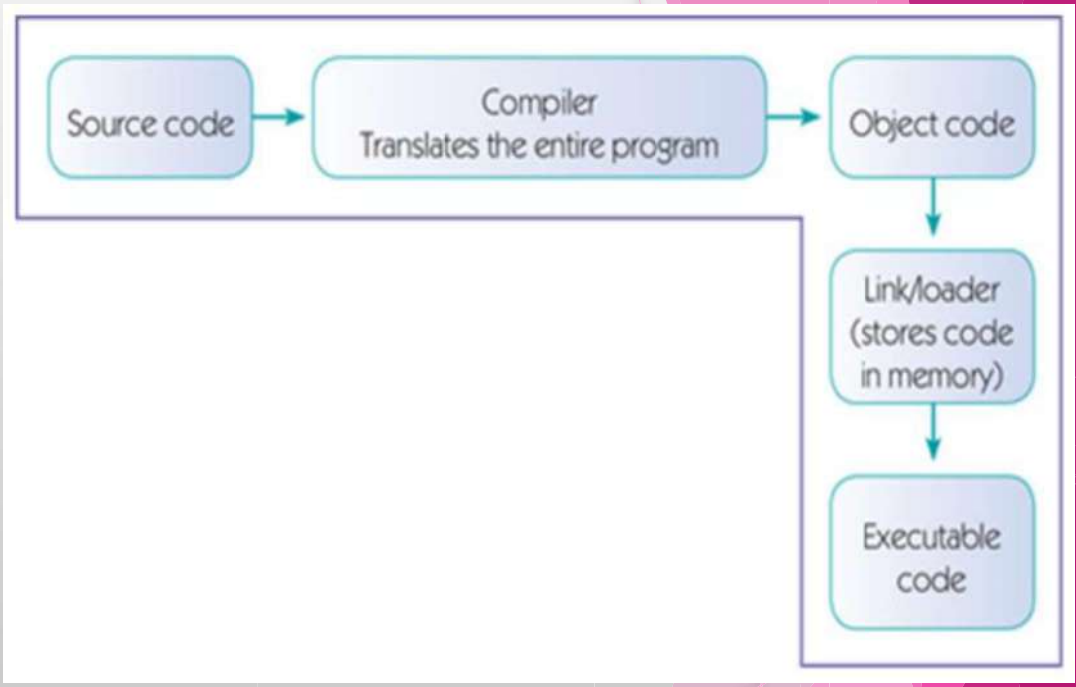


Compilers & Programs

- **Object program**
  - Output from the compiler
  - Equivalent machine language translation of the source program
  - Files usually have extension ‘.obj’
- **Executable program**
  - Output from linker/loader
  - Machine language program linked with necessary libraries & other files
  - Files usually have extension ‘.exe’

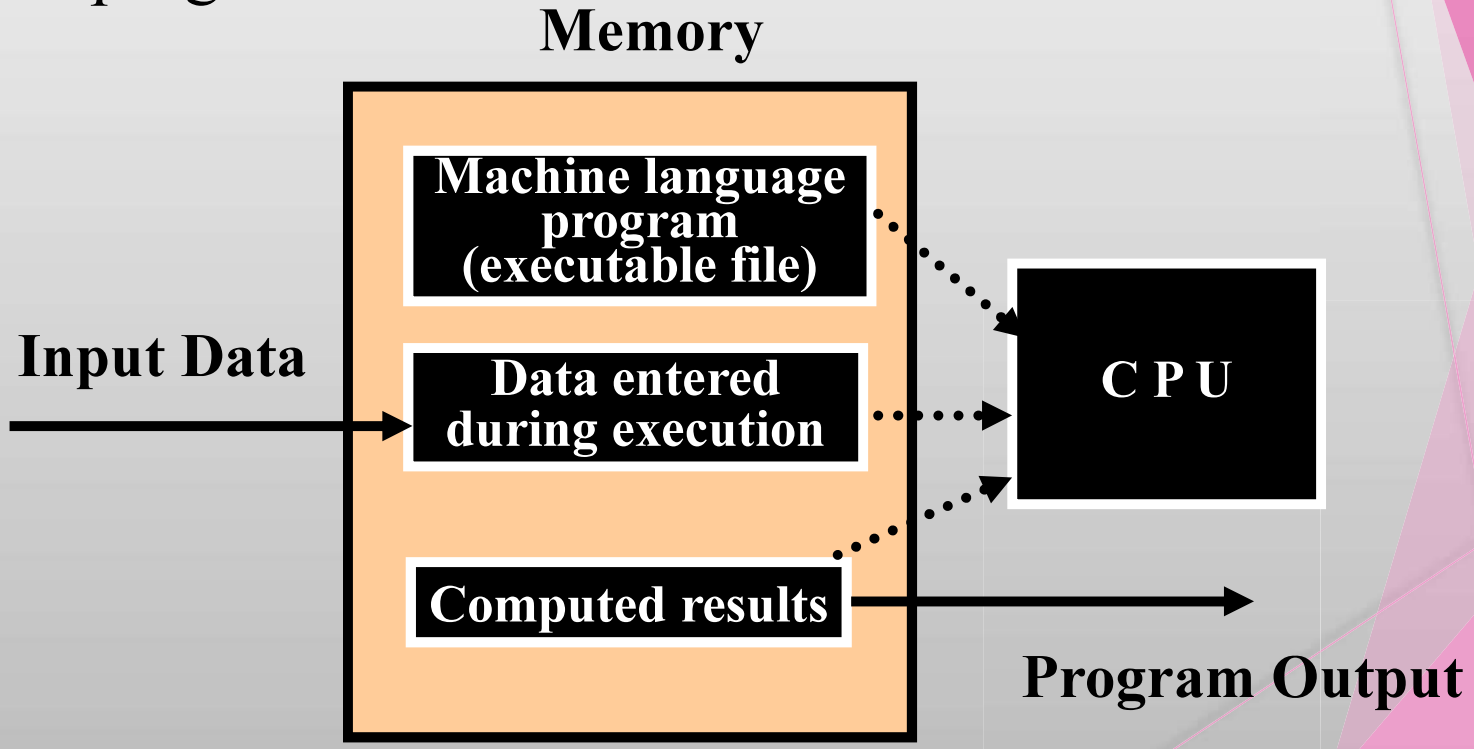
What is a Linker?

- A program that pulls other programs together so that they can run.
- Most programs are very large and consist of several modules.
- Even small programs use existing code provided by the programming environment called libraries.
- The linker pulls everything together, makes sure that references to other parts of the program (code) are resolved.



Running Programs

- Steps that the computer goes through to run a program:



Overview of C

- C is one of the most popular computer languages because it is structured, high-level, machine independent language.
- Allows software developers to develop programs without worrying about hardware platform.
- Key features of C:
  - Few keywords
  - Structures, unions – compound data types
  - Pointers – memory, arrays
  - External standard library – I/O, other facilities
  - Compiles to native code
  - Macro preprocessor