Problem Solving and Program Design - Chapter 1

Cory L. Strope

Overview of Computers and Programming

- Computer Hardware
- Computer Software
- Software Development (Problem Solving)
- Pseudocode
- Flowchart

Intro. to Computers

Computers receive, store, process, and output information.

Computer can deal with numbers, text, images, graphics, and sound, to name a few.

Computers are useless without programming.

Programming Languages allow us to write programs and thus communicate with computers.
It takes our code and converts it to binary (0 and 1) so the computer can understand it.

Different Types of Computers

- Personal Computer - used by everyday people, least powerful of the three and typically used by just one person at a time.
- Mainframes - used for real-time systems, ATMs, and such, very powerful and reliable computers.
- Supercomputers - used by research laboratories for computationally intensive applications such as weather forecasting, the largest capacity and fastest mainframes.

Hardware vs. Software

Hardware is the equipment used to perform the necessary computations.

i.e. CPU, monitor, keyboard, mouse, printer, etc.

Software is the programs that enable us to solve problems with a computers by providing it with a list of instructions to follow

i.e. Word, Internet Explorer, VI, etc.

Computer Hardware

- Main Memory
  - RAM - Random Access Memory - Memory that can be accessed in any order.
  - ROM - Read Only Memory - Memory that cannot be written to
- Secondary Memory - Hard disks, floppy disks, zip disks, CDs, & DVDs.
- Central Processing Unit (CPU) - Coordinating all computer operations and perform arithmetic and logical operations.
- Input/Output Devices - Monitor, printer, keyboard, & mouse.
- Computer Networks (not hardware, but configuration of the hardware) - WAN, LAN, MAN, Wireless-LAN.

Memory

Memory is an essential component in any computer.

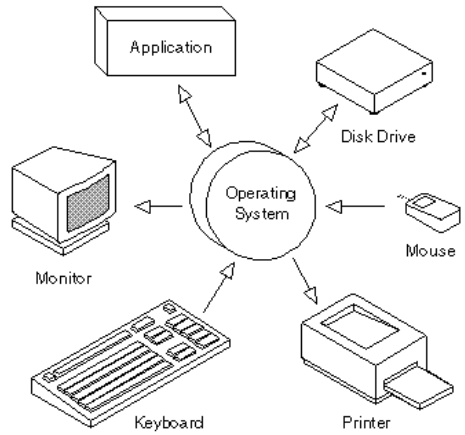
Anatomy of Memory

- Memory Cell - the storage locations
- Address - the location of the memory cell relative to other memory cells
- Content - what is stored in the memory cell
  - All programs run in memory
  - Every memory cell has content, whether we know it or not. So always initialize variables
- bit - deriving from **b**inary **d**ig**it** is either a 0 or 1.
- byte - a memory cell is actually a grouping of smaller units called bytes. A byte is made up of 8 bits.
  - This is about the amount of storage required to store a single character, such as the letter H.

Computer Software

Operating System - controls the interaction between machine and user.

- Communicate with computer user.
- Manage memory.
- Collect input/Display output.
- Read/Write data.



- Application Software - developed to assist a computer use in accomplishing specific tasks (i.e. Word, Excel, & Explorer).

Computer Languages

- Machine Language - A collection of binary numbers
  - Machine language is not standardized, and will vary between families of processors, such as Intel (x86) and Macintosh.
- Assembly Language - mnemonic codes rather than binary.
  - Low-level language - A language that is close to the hardware.
  - Same structure and set of commands as the hardware, but uses names instead of numbers.
- High-level Languages - combine algebraic expressions and symbols from English
  - High-level language (HLL) - Closer to human language, easier to read, write, and maintain.
  - Must be translated to Machine language
  - Independent from the hardware
  - (Fortran, Cobol, Lisp, **C**, Prolog, Pascal, C#, & Java).

Computer Languages

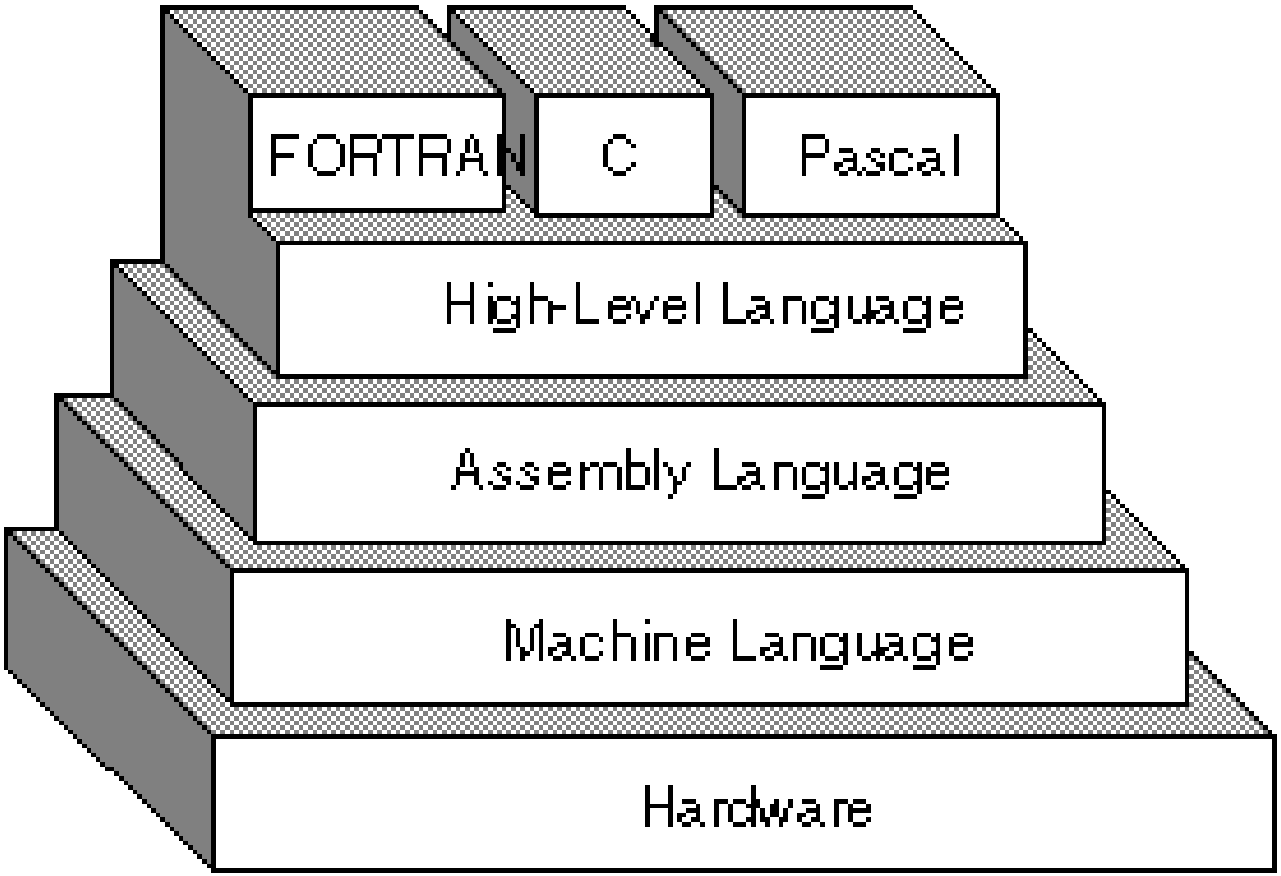


Image from
http://www.webopedia.com/TERM/H/high_level_language.html

Examples of Different Levels of Computer Languages

C source code:

```
char name[40];
printf("Please enter your name\n");
scanf("%s", name);
printf("Hello %s", name);
```

Assembly Code

```
push      offset string "Please enter your name\n"
(41364Ch)
call      dword ptr [__imp__printf (415194h)]
add       esp,4
lea       eax,[name]
push      eax
push      offset string "%s" (413648h)
call      dword ptr [__imp__scanf (41519Ch)]
add       esp,8
lea       eax,[name]
push      eax
push      offset string "Hello %s" (41363Ch)
call      dword ptr [__imp__printf (415194h)]
add       esp,8
```

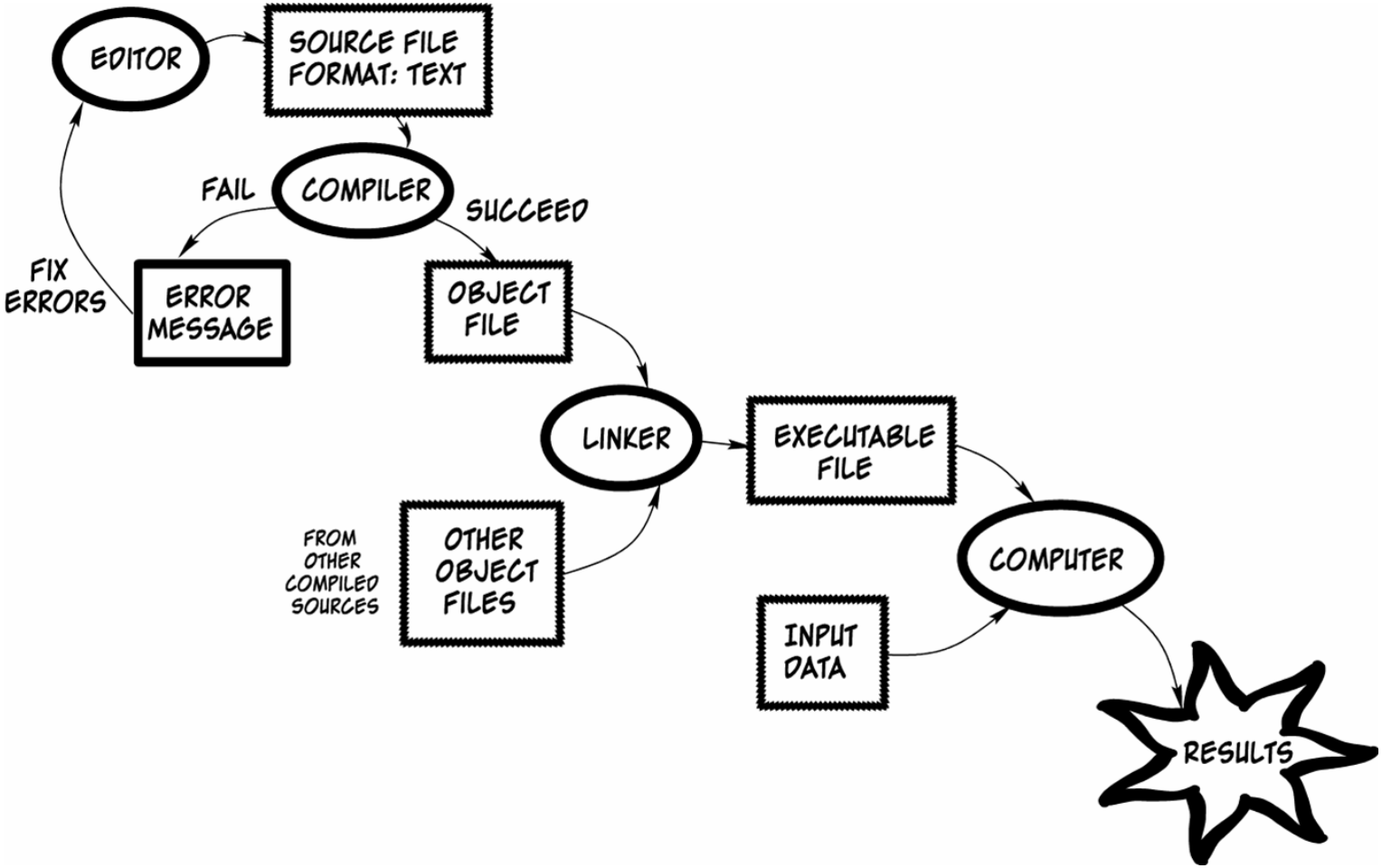
Machine Code:

```
68 4C 36 41 00 FF 15 94 51 41 00 83 C4 04 8D 45 D8
50 68 48 36 41 00 FF 15 9C 51 41 00 83 C4 08 8D 45
D8 50 68 3C 36 41 00 FF 15 94 51 41 00 83 C4 08
```

Compiling Code

- **Compiling** is the process of taking your source code and turning it into executable code.
- **Source file** - A file containing the program code.
  - A compiler turns the source file into an object file.
- **Object file** - a file containing machine language instructions.
  - A **Linker** turns the object file into an executable.
- **Integrated Development Environment (IDE)** - a program that combines simple word processing with a compiler, linker, and loader.

Compiler Overview - Page 18



Software Development Method

- Specify the problem requirements.
- Analyze the problem.
- Design the algorithm to solve the problem.
- Implement the algorithm.
- Test and verify the completed program.
- Maintain and update the program.

Steps Defined

- Problem - specifying the problem requirements forces you better understand the problem.
- Analysis - analyzing the problem involves identifying the problems inputs, outputs, and addition requirements.
- Design - designing the algorithm to solve the problem requires you to develop a list of steps that solve the problem and verify the steps.
- Implementation - implementing is writing the algorithm as a program.
- Testing - testing accuracy of the program.
- Maintenance - maintaining involves finding previously undetected errors and update the program to code.

Failure is part of the process.

Example: Converting Miles to Kilometers

- Problem - you summer job wants you to convert a list of miles to kilometers
- Analysis
  - Input: Number of miles,
  - Output: Number of kilometers,
  - Relevant info: 1 mile = 1.609 kilometers
  - Design:
    1. Get distance in miles
    2. Convert to kilometers
    3. Display kilometers

Implementation

```
#include <stdio.h>
int main(void)
{
    double miles, kilometers;
    printf("How many miles do you have?");

    scanf("%lf",&miles);

    kilometers = miles * 1.609;
    printf("You have %f kilometers\n",kilometers);

    return 0;
}
```

Testing

We need to test the previous program to make sure it works. To test we run our program and enter different values and make sure the output is correct.

Pseudocode

- **Pseudocode** - A combination of English phrases and C constructs to describe algorithm steps.
- **Flowchart** - A diagram that shows the step-by-step execution of a control structure.
  - Less commonly used than pseudocode, but gives you a visual feel for the flow of the program.
- **Algorithm** - A list of steps for solving a problem.

Pseudocode

- Pseudocode is simply an outline of a program.
  - Cannot be compiled nor executed,
  - There are no formatting or syntax rules.
- The benefit of pseudocode is that it enables the programmer to concentrate on the algorithms without worrying about the syntactic details of a particular programming language. In fact, You can write pseudocode without even knowing what programming language you will use for the final implementation.
- Program M2KM:
  1. Input Miles
  2. kilometers $= 1.609 \times$ miles
  3. Output Miles

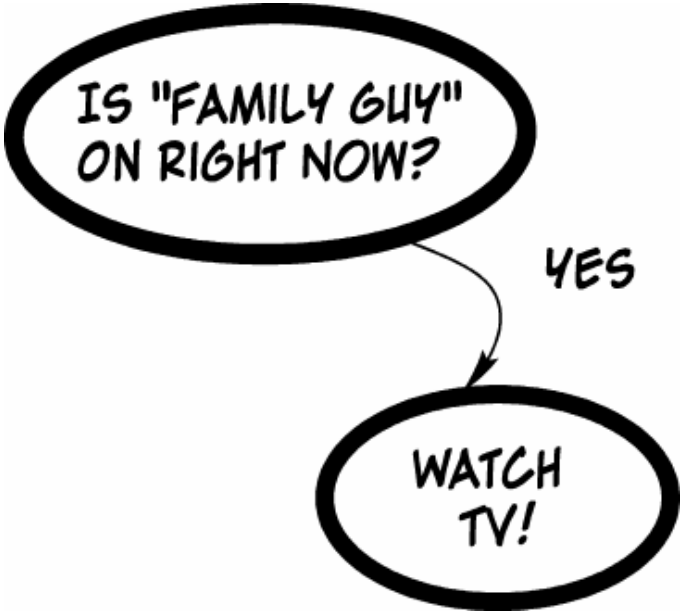
Seven Structures

- **Control Structure:** A method for controlling the order in which instructions execute.
- In C, there are 7 control structures
  1. Sequential
  2. If Then
  3. If Then Else
  4. Switch
  5. For Do
  6. While Do
  7. Do While

Sequential

- Use sequential structure whenever program statements follow one after the other with no decisions and no repetitions.
- Processing flow is always downward from top to bottom in sequential structures.

If Then

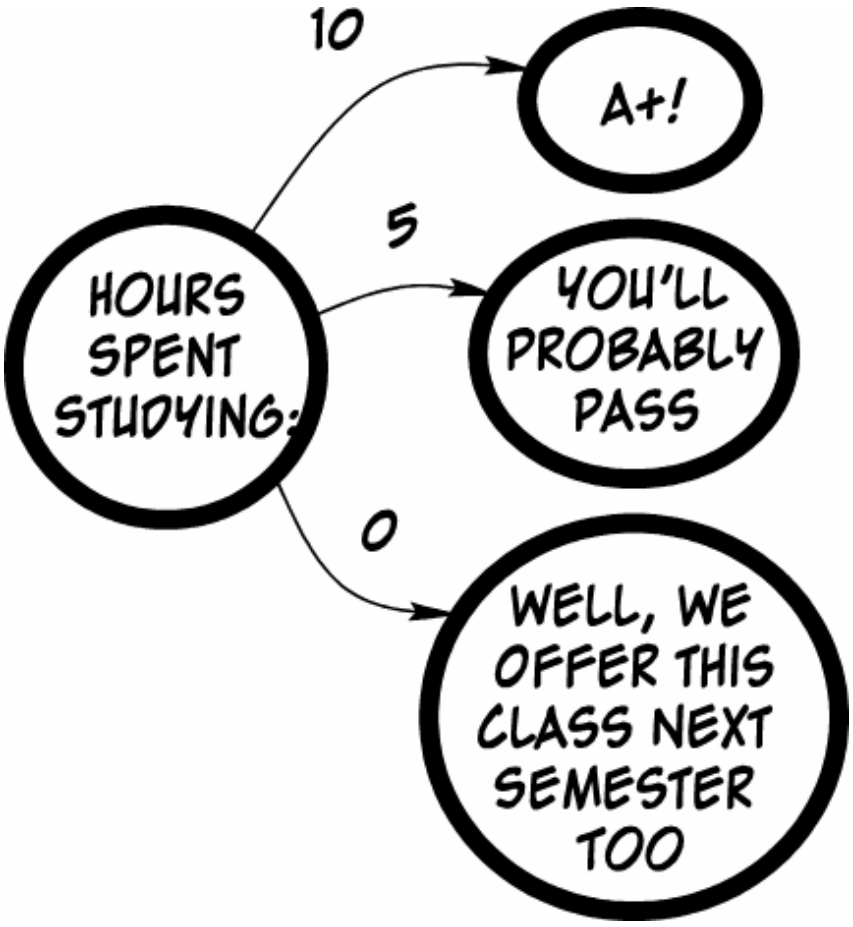


- Use the If Then structure when there is a single process to do or not do.
- Processing flow is down either the left side or the right side.

If Then Else

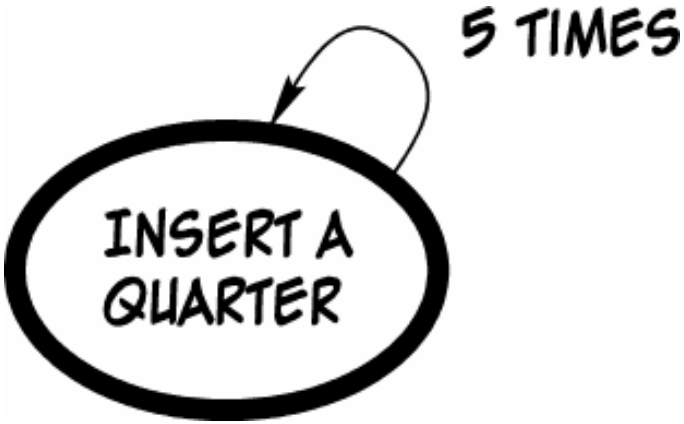
- Use If Then Else when one of two processes must be chosen.
- Processing flow is down either the left side or the right side.

Switch



- Whenever there are multiple potential options depending on a single values, use the switch statement.
- Example: Multiple If-Then-Else statements.

For Do

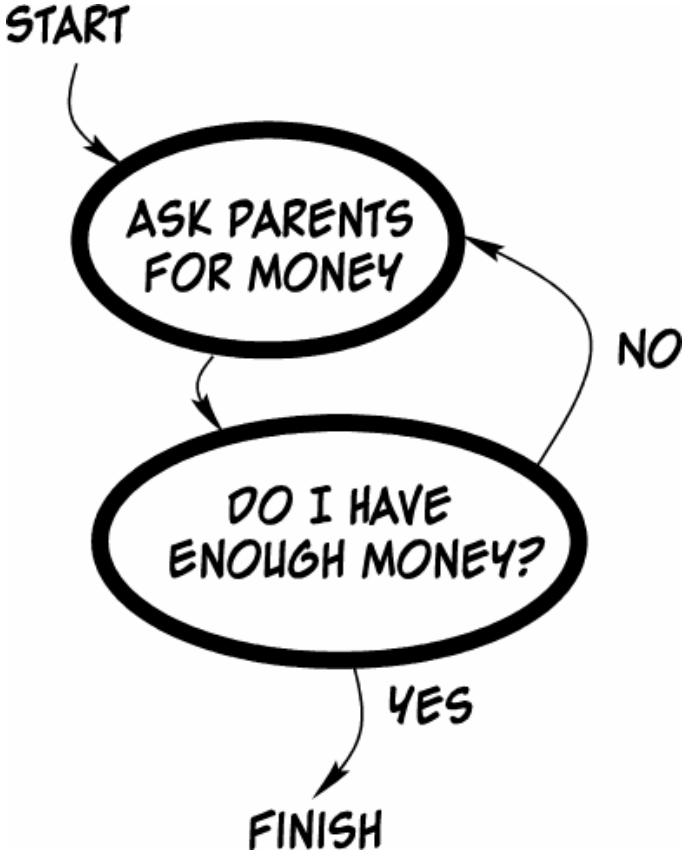


- Use For Do when you need to repeat an action multiple times, and you know how many times you will repeat it.

While Do

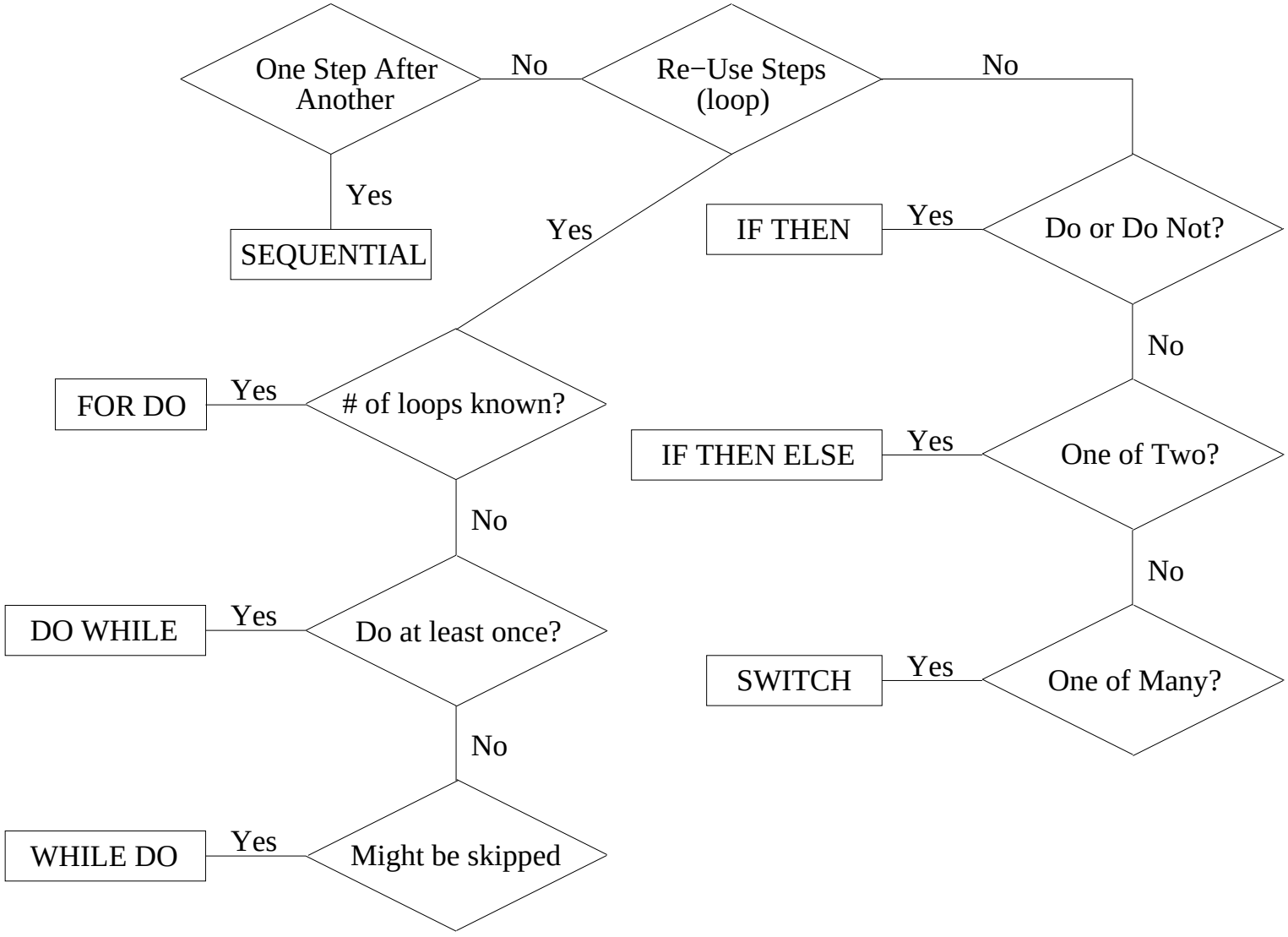
- Use While Do when the number of loops is unknown and process might not be executed at all (indeterminate pre-test).

Do While

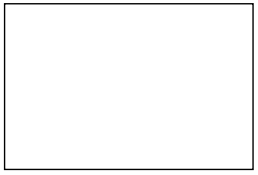


- Use Do While when number of loops is unknown and process must be done *at least once* (indeterminate post-test).

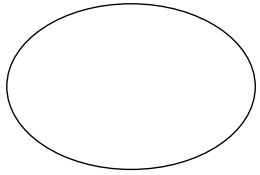
Which Structure to Use



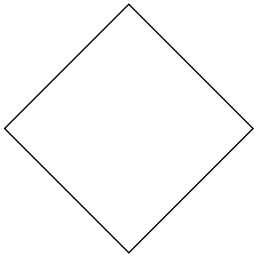
Flowchart



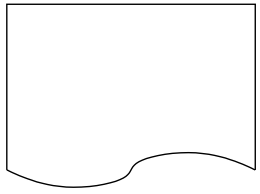
Action to perform



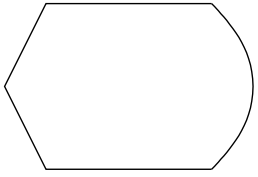
Start or Terminal



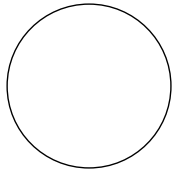
Decision to be made



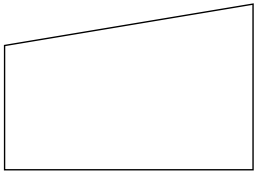
Document



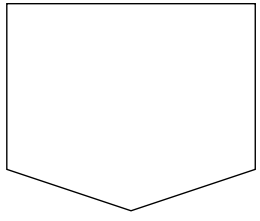
Display information



On page connector

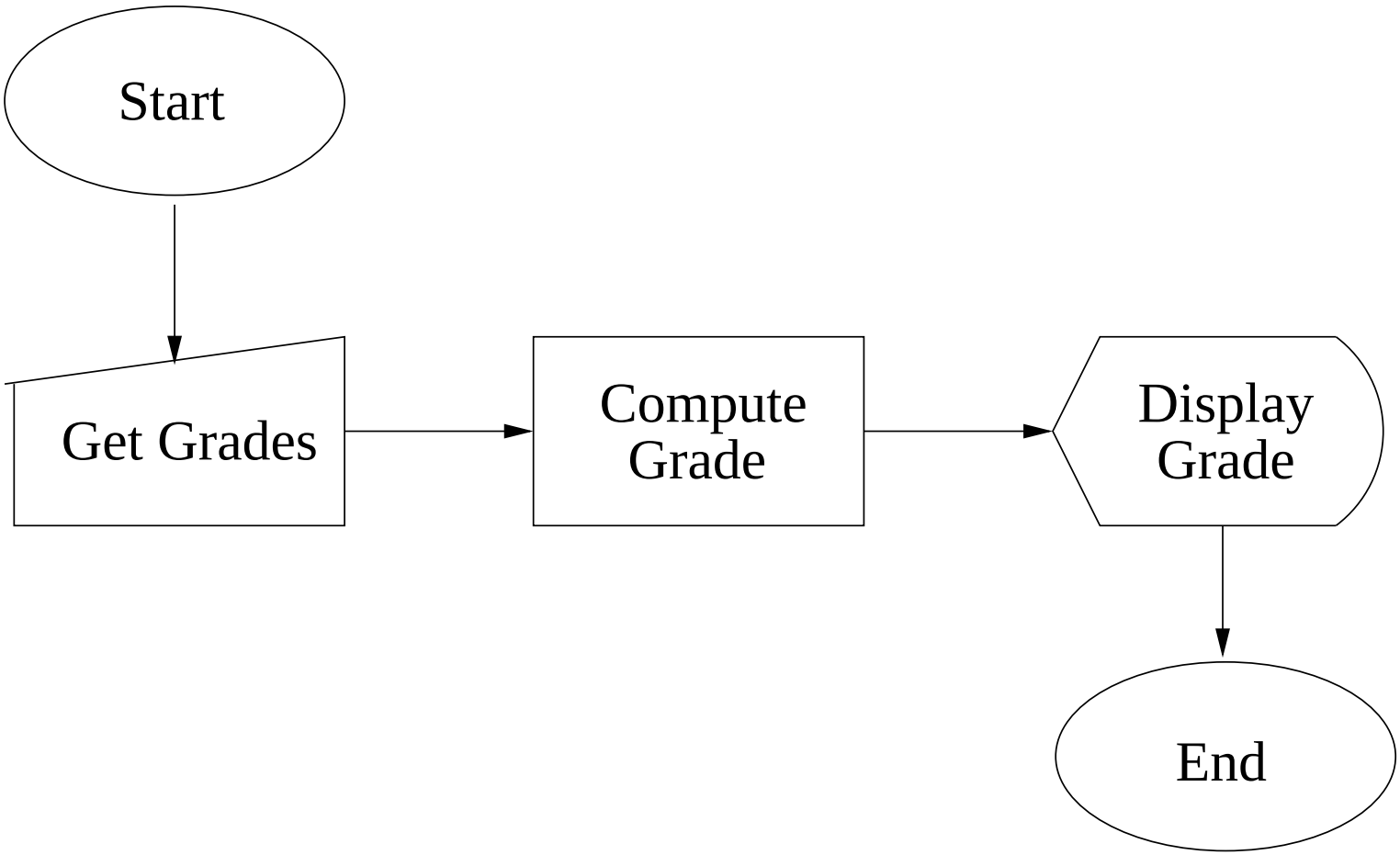


Manual input



Off page connector

Example of Flowchart



Example of Pseudocode

Problem - How do I compute my grade for this class?

- **Specify the problem** - get the grades for the class and compute the final grade.
- **Analyze the problem** - we need to input the grades, output the grade, and percentage for each part of the class.
- **Design** -
  1. Get the grades homeworks, quizzes, exams, lab, and CodeLab
  2. Grade = homework * .20 + quizzes * .10 + CodeLab * .10 + exam1 * .25 + exam2 * .25 + lab * .10
  3. Output the Grade
- Implement - We can implement after we learn how to program.

Questions over chapter 1?