Structured Programming

Mohamed Khedr

Lecture 1

Outline

1. Course Information
2. Overview of C Programming
3. My first program : Hello, World!

Course Outline

- **An introduction to computer programming**
- **Problem solving skills and software development methods**
- **Data types, operators and simple functions**
- **Selection structure (IF and Switch statements)**
- **Repetition and loop statements**
- **Function & modular programming**
- **Arrays & applications**
- **Multidimensional arrays**

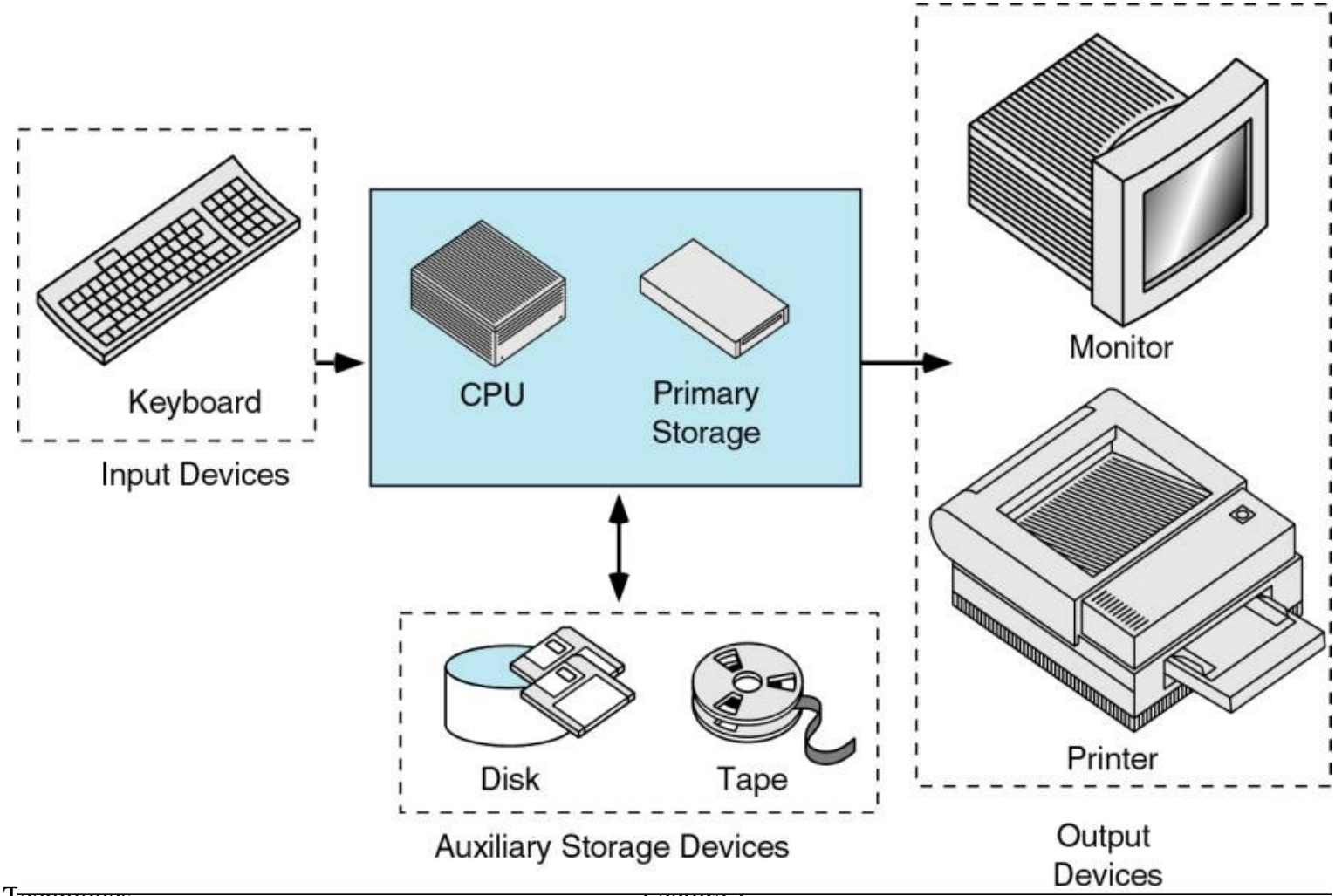
Lecture One Outline

- **Overview of Computer, Programming and Problem Solving**
- **What is Programming?**
- **Programming Life-Cycle Phases**
- **Algorithm Basic Control Structures**
- **Sample problem**

Background
What is a Computer?

- **Device capable of performing computations and enacting logical decisions**
- **Perform these tasks millions and even billions of times faster than people can**
- **Computers process data through sets of instructions called *computer programs***
- **Programs guide the computer through actions as specified by people called *computer programmers*; For this course, that will be YOU.**

Background
Computer Components



What is Programming?

- **Given a well-defined problem:**

1. Find an algorithm to solve a problem.
2. Express that algorithm in a way that the computer can execute it.

Programming Life Cycle Phases

1. **Analyze the problem.**
   - **This involves identifying the data you have to work with it, the desired results, and any additional requirements or constrains on the solution.**

2. **Design the algorithm to solve the problem.**
   - **An algorithm is a step-by-step procedure for solving a problem in a finite amount of time.**

3. **Implement the algorithm.**
   - **Each algorithm is converted into one or more steps in a programming language. This process is called CODING.**

Programming Life Cycle Phases

4. Test and verify the completed program.

- **Run the program several times using different sets of data, making sure that it works correctly for every situation in the algorithm .**

- **if it does not work correctly, then you must find out what is wrong with your program or algorithm and fix it--this is called DEBUGGING**

5. Maintain and update the program.

- **maintenance begins when your program is put into use and accounts for the majority of effort on most programs.**

- **MODIFY the program to meet changing requirements or correct errors that show up in using it.**

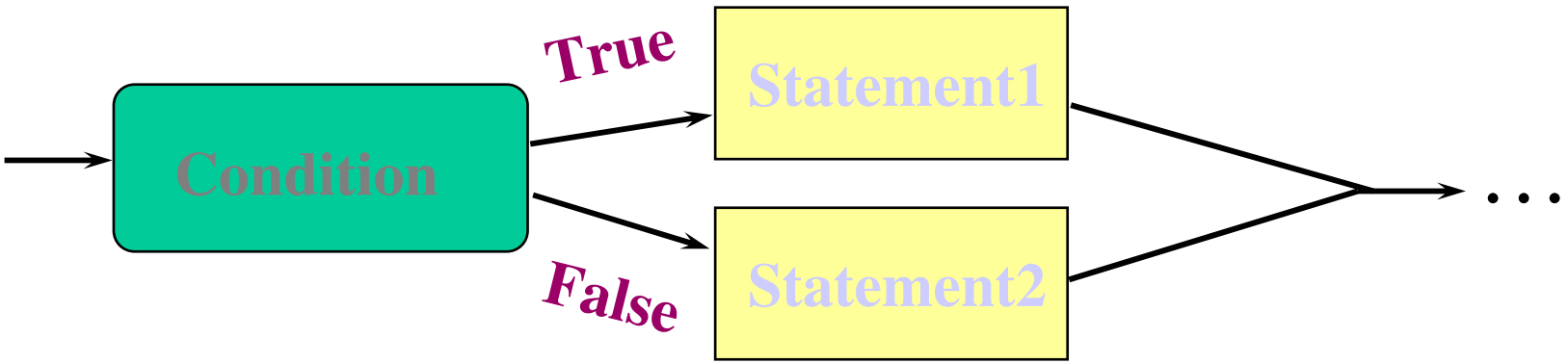
Algorithm Basic Control Structures

- **a sequence is a series of statements that execute one after another**
- **selection (branch) is used to execute different statements depending on certain conditions**
- **Looping (repetition) is used to repeat statements while certain conditions are met.**
- **a subprogram is used to break the program into smaller units**

Control Structures
Sequences

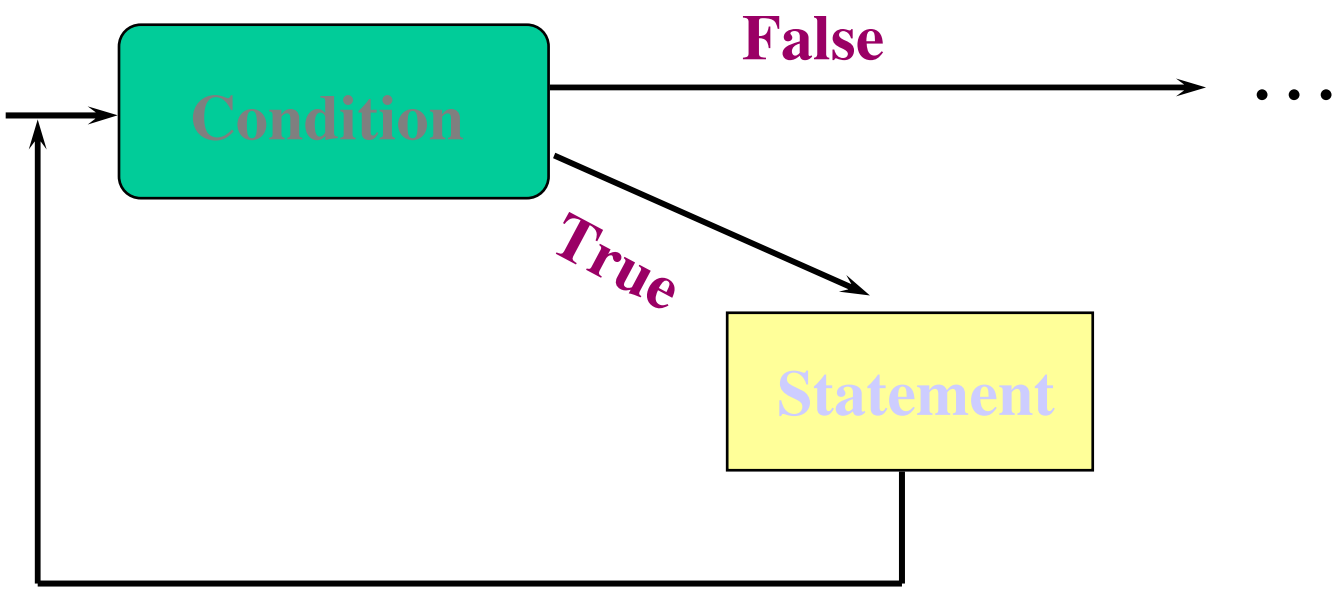
Control structures
Selection (Branching)

IF Condition THEN Statement1 ELSE Statement2

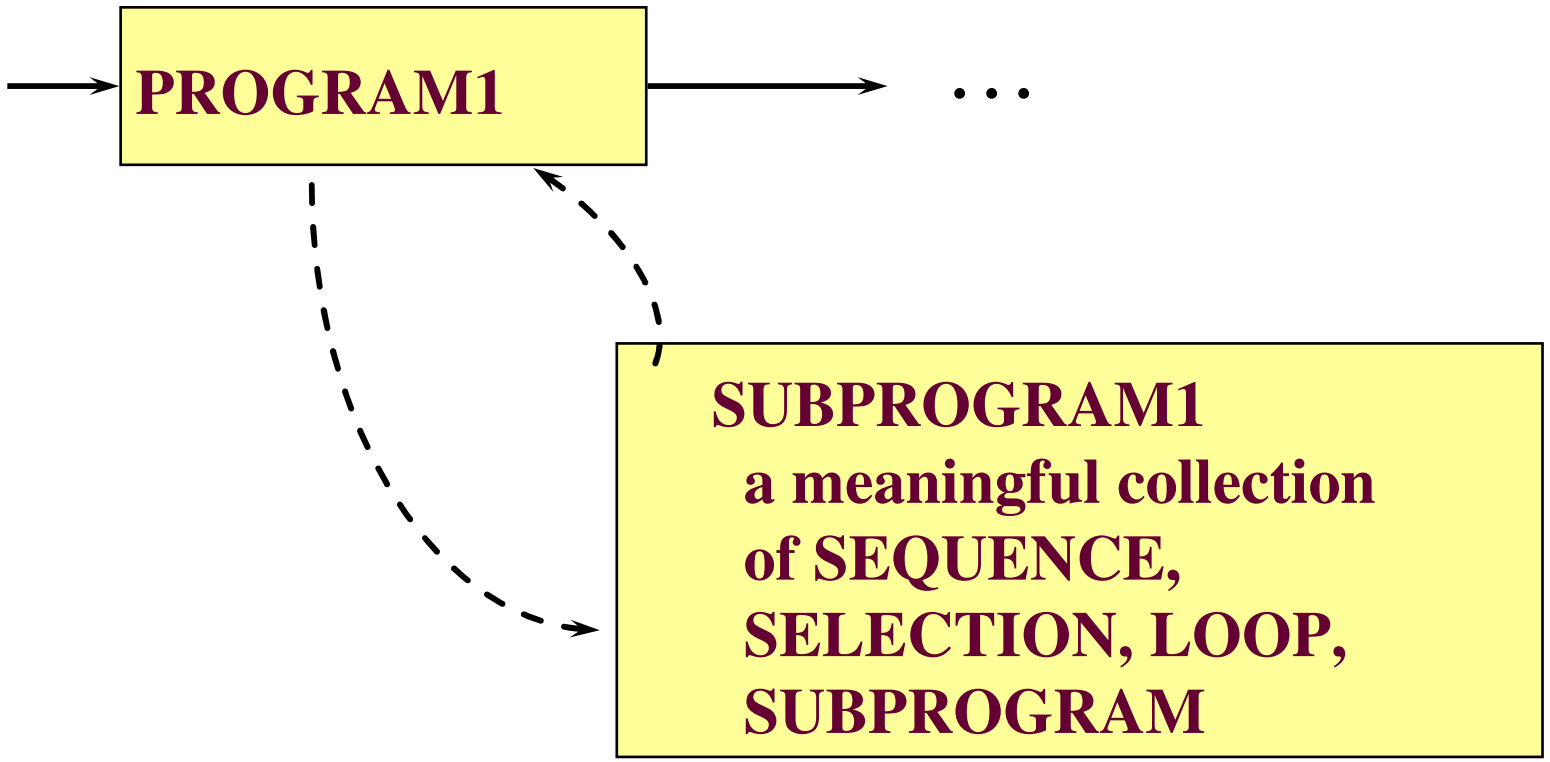


Control structures
Loop (Repetition)

WHILE Condition Statement1



Control structures
Subprogram (Function)



Programming Languages

Three types of programming languages

1. Machine languages
   - Strings of numbers giving machine specific instructions
   - Example:

```
+1300042774
+1400593419
+1200274027
```

2. Assembly languages
   - English-like abbreviations representing elementary computer operations (translated via assemblers)
   - Example:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```

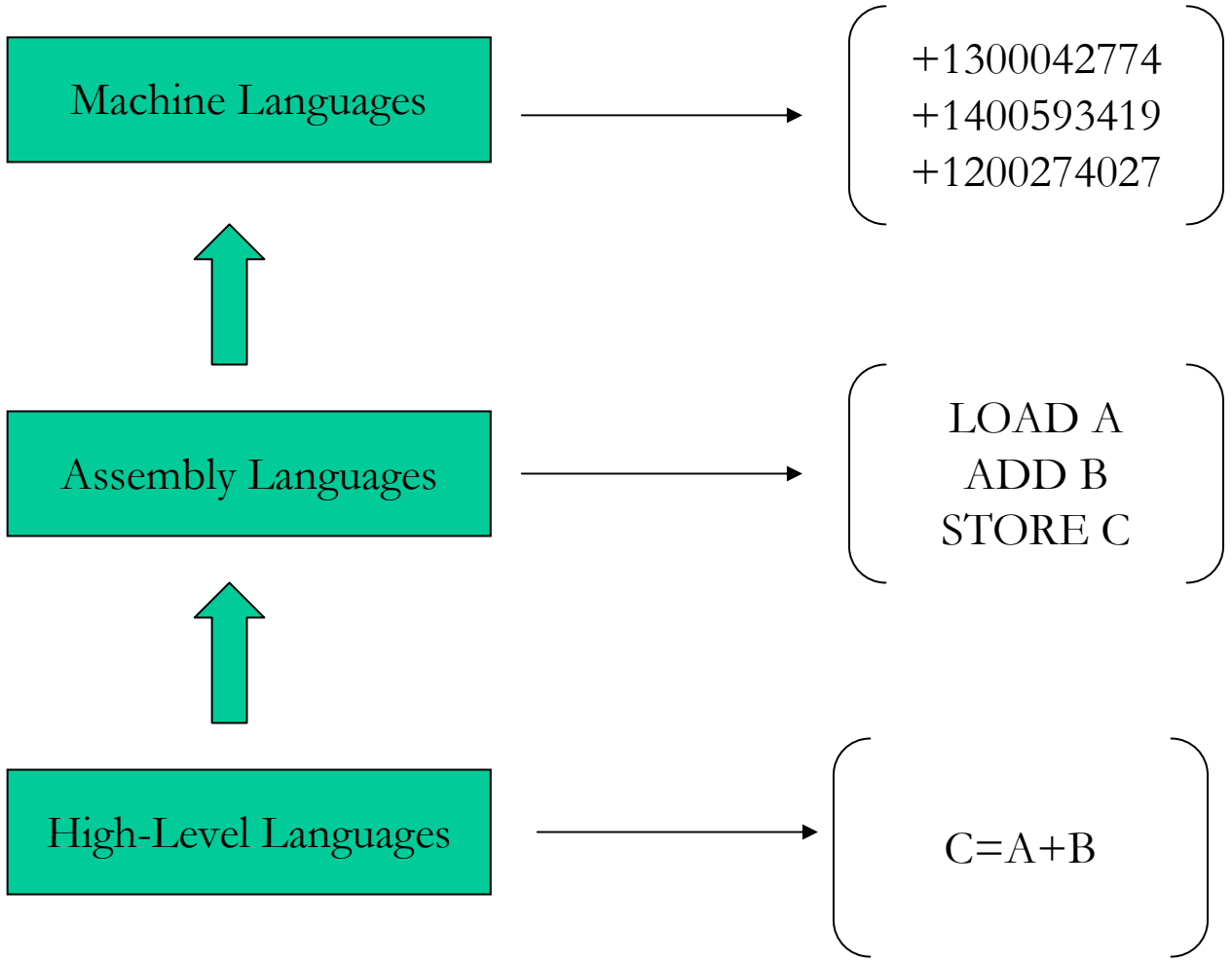
Programming Languages (2)

Three types of programming languages (continued)

3. High-level languages
   - Codes similar to everyday English
   - Use mathematical notations (translated via compilers)
   - Example:

```
grossPay = basePay + overTimePay
```

Programming Languages (3)



History of C

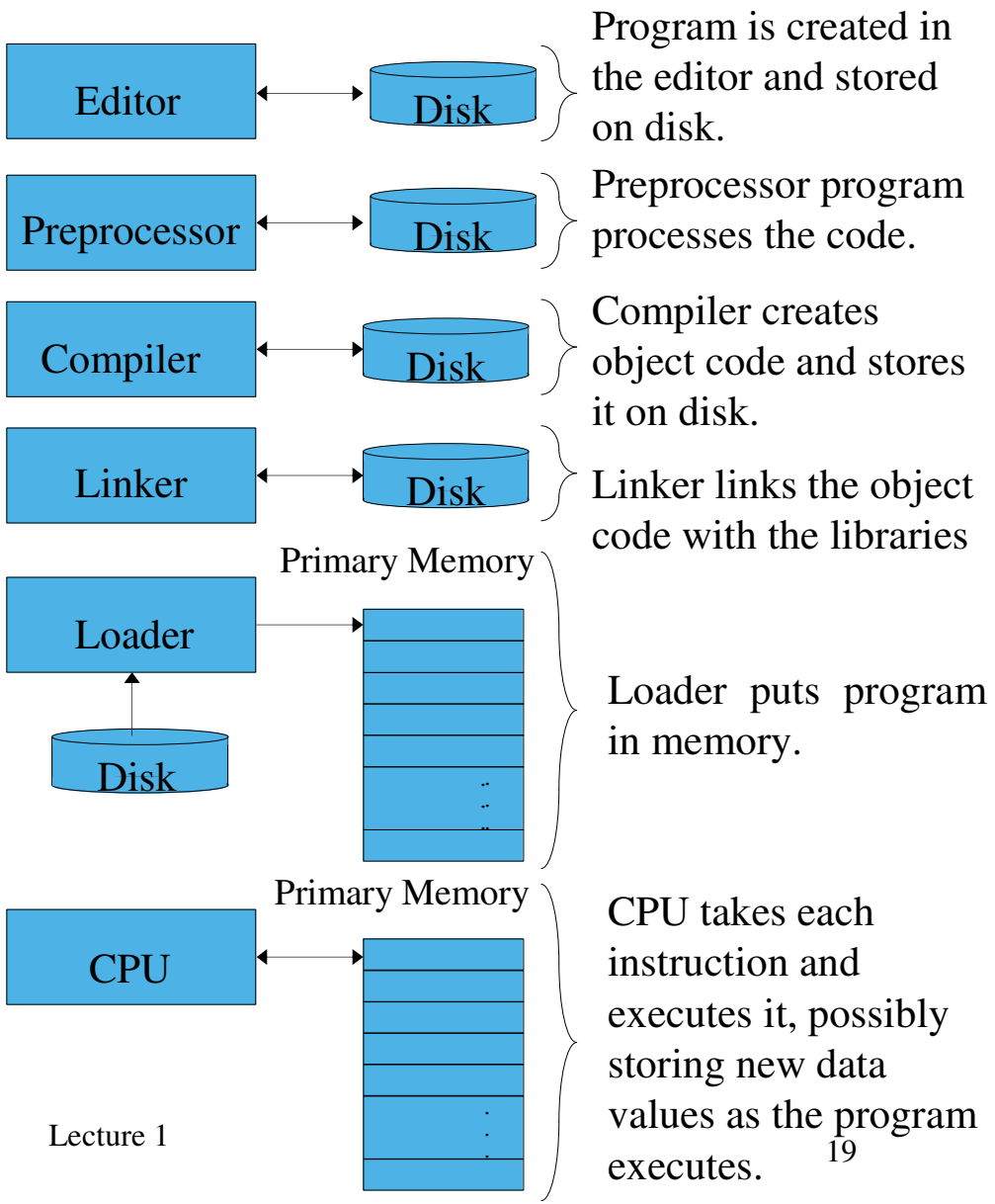
- C
  - Evolved by Ritchie from two previous programming languages, BCPL and B
  - Used to develop UNIX
  - Used to write modern operating systems
  - Hardware independent (portable)
  - By late 1970's C had evolved to "Traditional C"
- Standardization
  - Many slight variations of C existed, and were incompatible
  - Committee formed to create a "unambiguous, machine-independent" definition
  - Standard created in 1989, updated in 1999

The C Standard Library

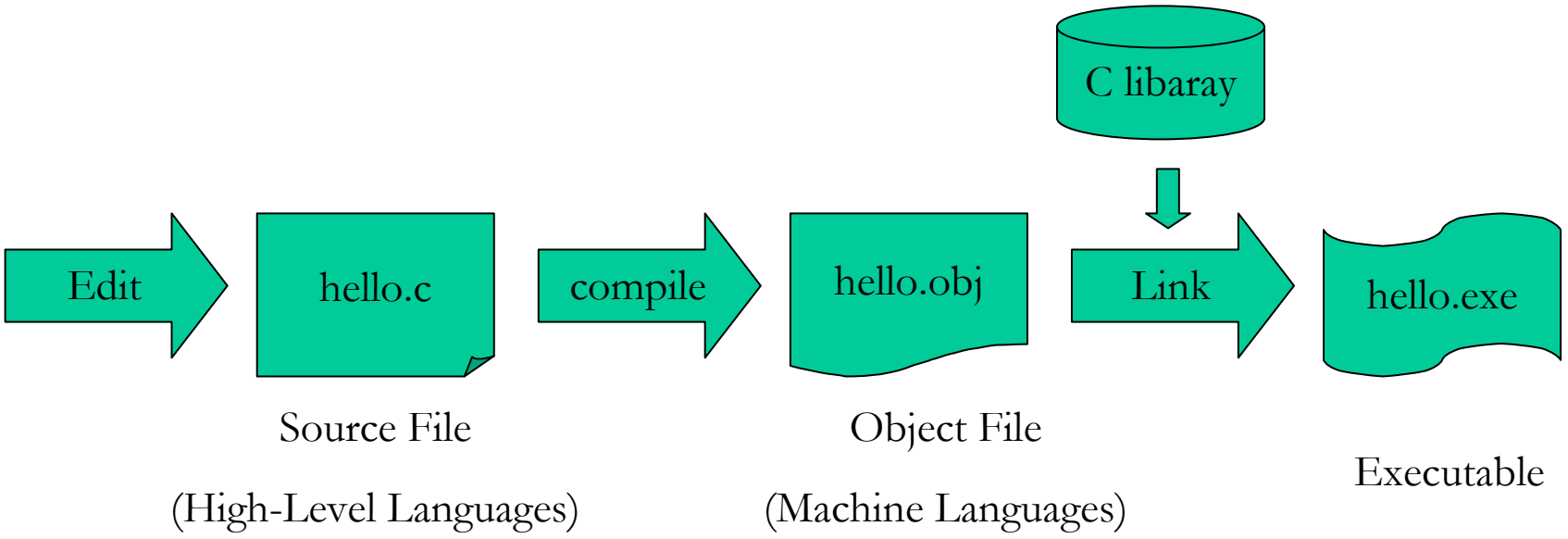
- C programs consist of pieces/modules called functions
  - A programmer can create his own functions
    - Advantage: the programmer knows exactly how it works
    - Disadvantage: time consuming
  - Programmers will often use the C library functions
    - Use these as building blocks
  - Avoid re-inventing the wheel
    - If a premade function exists, generally best to use it rather than write your own
    - Library functions carefully written, efficient, and portable

Basics of a Typical C Program Development Environment

- Phases of C Programs:
  1. *Edit*
  2. *Preprocess*
  3. *Compile*
  4. *Link*
  5. *Load*
  6. *Execute*



Creating Programs



My First C Program: Hello World!

```
/* My first C program */


#include <stdio.h>

int main()
{
   printf("Hello World¥n");
   return 0;
}
```

Comments

/* My first C program */

- Comments are a way of explaining what a program does.
- They are put after // or between /* */.
- Comments are ignored by the compiler and are used by you and other people to understand your code.
- You should always put a comment at the top of a program that tells you what the program does because one day if you come back and look at a program you might not be able to understand what it does but the comment will tell you.
- You can also use comments in between your code to explain a piece of code that is very complex.
- Here is an example of how to comment the Hello World program:

Updated Comments

```
/* Author: Tim Ji
   Date: 1/10/2005
   Description: Writes the words "Hello World"
on the screen
*/

#include<stdio.h>

int main()
{
   printf("Hello World¥n"); //prints "Hello World"
   return 0;
}
```

Preprocessing

#include <stdio.h>

- This includes a file called stdio.h which lets us use certain commands.
- stdio is short for Standard Input/Output which means it has commands for input like reading from the keyboard and output like printing things on the screen.

Program Entry Point

int main()

- **int** is what is called the return value which will be explained later on in this course.
- **main** is the name of the point where the program starts and the brackets are there for a reason that you will learn in the future but they have to be there.

Blocks

{}

- The 2 curly brackets are used to group all the commands together so it is known that the commands belong to main.
- These curly brackets are used very often in C to group things together.

Functions

printf("Hello World\n");

- This is the **printf** command and it prints text on the screen.
- The data that is to be printed is put inside brackets. You will also notice that the words are inside quotation marks because they are what is called a string.
- Each letter is called a **character** and a series of characters that is grouped together is called a **string**.
- Strings must always be put between quotation marks. The \n is called an escape sequence and represents a newline character and is used because when you press ENTER it doesn't insert a new line character but instead takes you onto the next line in the text editor.
- You have to put a semi-colon after every command to show that it is the end of the command.

Return from Function

return 0;

- The int in int main() is short for **integer** which is another word for number.
- We need to use the return command to return the value 0 to the operating system to tell it that there were no errors while the program was running.
- Notice that it is a command so it also has to have a semi-colon after it.

Programming Style

- Indentation
  - You will see that the **printf** and **return** commands have been **indented** or moved away from the left side.
  - This is used to make the code more readable.
  - It seems like a stupid thing to do because it just wastes time but when you start writing longer, more complex programs, you will understand why indentation is needed.

Variables

```
int x, y, z;
```

- Variables declared at beginning of a function
  - Before any executable statements
- Number of bytes for each data type
  - char = 1 (-128 to 127)
  - short = 2 (-32768 to 32767)
  - int & long = 4 (-2,147,483,648 to +2,147,483,647)
  - float = 4 ($\pm \sim 10^{-44}$ to $\sim 10^{38}$)
  - double = 8 ($\pm \sim 10^{-323}$ to $\sim 10^{308}$)
- Also have unsigned char, short, int, etc.

Variable Names

- Restrictions
  - Made up of letters & digits
  - 1st character must be a letter
  - Underscore ("_") counts as a letter
    - Often used in library routines
  - Case sensitive
    - APPLE & apple are different variables
  - Less than 31 characters
  - Cannot use reserved keywords
    - auto, break, case, …, void, volatile, while

Arithmetic Operators

1. ( )
   - Parenthesis (highest precedence)
2. *, /, %
   - Multiplication, division, modulus
   - Evaluated left to right
3. +, -
   - Addition and subtraction
   - Evaluated left to right

Arithmetic Operators

- Integer division produces an integer result
  - 1 / 2 evaluates to 0 (no rounding up!)
  - 19 / 5 evaluates to 3 (no rounding up!)
- Modulus (%) = remainder after division
  - 1 % 2 evaluates to 1
  - 17 % 5 evaluates to 2
- Implicit conversion – if an operation has operands of different types, the "narrower" one will be converted to the "wider" one
  - 2.0 / 5 evaluates to 0.4 (a float)

Equality, Relational Operators

- Equality operators
  - == (equal) – common error: = instead of ==
  - != (not equal)
- Relational operators
  - > (greater than)
  - < (less than)
  - >= (greater than or equal)
  - <= (less than or equal)
- Will return a 0 (false) or 1(true)

if Control Structure

- A program can make a decision using the if control structure and the equality or relational operators

```
#include <stdio.h>
int main(void)
{
    if(1 > 2) printf("%d > %d\n",1,2);
    if(1 <= 2) printf("%d <= %d\n",1,2);
    return 0;
}
```

Input in C

```
#include <stdio.h>
int main(void){
    int x = 0;
    printf("Enter an integer: ");
    scanf("%d", &x);
    printf("The integer is %d\n", x);
    return 0;
}
```

scanf Function

- `scanf("%d", &x);`
  - First argument: format control string
    - Indicates type of data to be entered
  - Second argument: location in memory
    - Use an ampersand (&) to give the address where the variable is stored
  - The computer will wait for the user to enter a value & push the enter key

Class Exercise 2

- Write a program that inputs two integers, adds them together, and determines if the result is a multiple of 3 (evenly divisible by 3)