

# CMPE 252

# C PROGRAMMING

---

SPRING 2021

WEEK 1-2

# Instructors, TAs

- Instructors

- Asst. Prof. Dr. Aslı Gençtav (Section 1&2)
  - [asli.genctav@tedu.edu.tr](mailto:asli.genctav@tedu.edu.tr)
  - Office hours: Thursday 13.00-14.00
- Assoc. Prof. Dr. Gökçe Nur Yılmaz (Section 3&4)
  - [gokce.yilmaz@tedu.edu.tr](mailto:gokce.yilmaz@tedu.edu.tr)
- Assoc. Prof. Dr. Kasım Öztoprak (Section 5&6)

- TAs

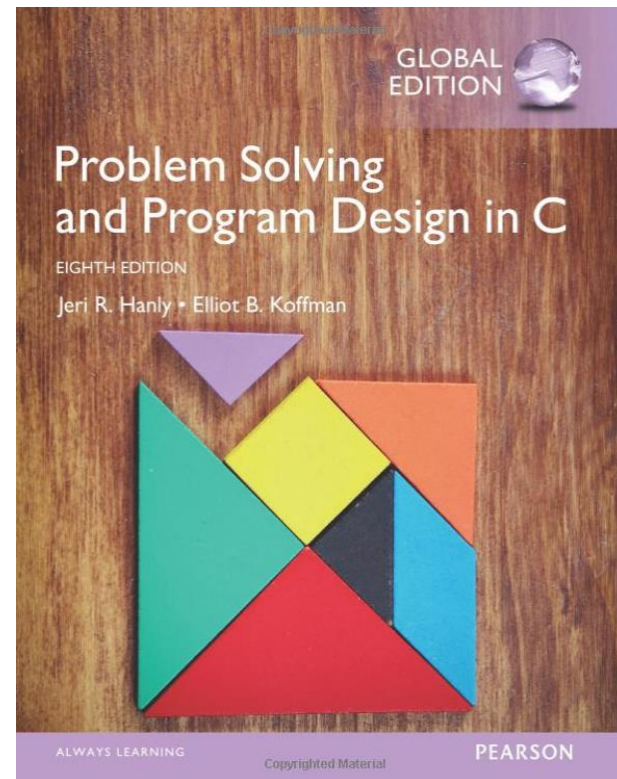
- Bedrettin Çetinkaya
  - [bedrettin.cetinkaya@tedu.edu.tr](mailto:bedrettin.cetinkaya@tedu.edu.tr)
- Mehmet Bahadır Aşkın
  - [bahadir.askin@tedu.edu.tr](mailto:bahadir.askin@tedu.edu.tr)
- Nigar Ercan
  - [nigar.ercan@tedu.edu.tr](mailto:nigar.ercan@tedu.edu.tr)

# Schedule

- Section 1: Monday 11-13 / *Wednesday 13-15*
- Section 2: Wednesday 11-13 / *Friday 9-11*
- Section 3: Tuesday 13-15 / *Friday 9-11*
- Section 4: Tuesday 15-17 / *Wednesday 15-17*
- Section 5: Tuesday 13-15 / *Friday 11-13*
- Section 6: Tuesday 15-17 / *Friday 13-15*

# Textbook

- «Problem Solving & Program Design in C» by Jeri R. Hanly & Elliot B. Koffman, Eighth Edition, Global Edition



# Grading

- Laboratory Works [20%]
  - There will be 6 graded lab works.
  - The highest 5 lab grades will be considered (the lowest one will be discarded).
  - Each lab is worth 4 points.
- Homework [24%]
  - There will be 3 homework.
  - Each homework is worth 8 points.
- Quizzes [15%]
  - There will be 3 quizzes.
  - Each quiz is worth 5 points.
  - Quizzes will be held on Saturday.
- Midterm Exam [16%]
- Final Exam [25%]

# Academic Integrity

- <https://student.tedu.edu.tr/tr/student/akademik-durustluk-ilkeleri>
- [https://www.tedu.edu.tr/sites/default/files/content\\_files/docs/Yonergeler/akademik\\_durustluk\\_ilkeleri\\_ihlalinde\\_uygulanacak\\_usul\\_ve\\_esaslar.pdf](https://www.tedu.edu.tr/sites/default/files/content_files/docs/Yonergeler/akademik_durustluk_ilkeleri_ihlalinde_uygulanacak_usul_ve_esaslar.pdf)
- [https://student.tedu.edu.tr/sites/default/files/content\\_files/docs/Yonergeler/ogrenciler\\_icin\\_akademik\\_durustluk\\_belgesi.pdf](https://student.tedu.edu.tr/sites/default/files/content_files/docs/Yonergeler/ogrenciler_icin_akademik_durustluk_belgesi.pdf)

# Attendance

- Attendance to the classes and labs are mandatory.
- Any student who misses more than 1 lab session will get FX.
- Any student who misses more than 8 lecture hours will get FX.
- Attendance to the lectures will be taken based on
  - Attendance logs in Zoom meeting reports
  - Answers to poll questions during the lectures

# Communication via Moodle

- Announcements
- Lecture notes
- Homeworks
- Quizzes
- Labs



- The lectures will be recorded and uploaded to Microsoft Stream.
  - For students who are not able to attend the classes online for a valid reason such as being abroad, health problems, and etc.

- Computer/Internet Quota problems
  - Send e-mail to Registrar's Office ([yim@tedu.edu.tr](mailto:yim@tedu.edu.tr))

# Outline

- **Syllabus**
- Overview of Computers and Programming, Chapter 1
  - Importance of learning multiple programming languages
- Overview of C, Chapter 2

# Syllabus

- What topics we will cover throughout this semester

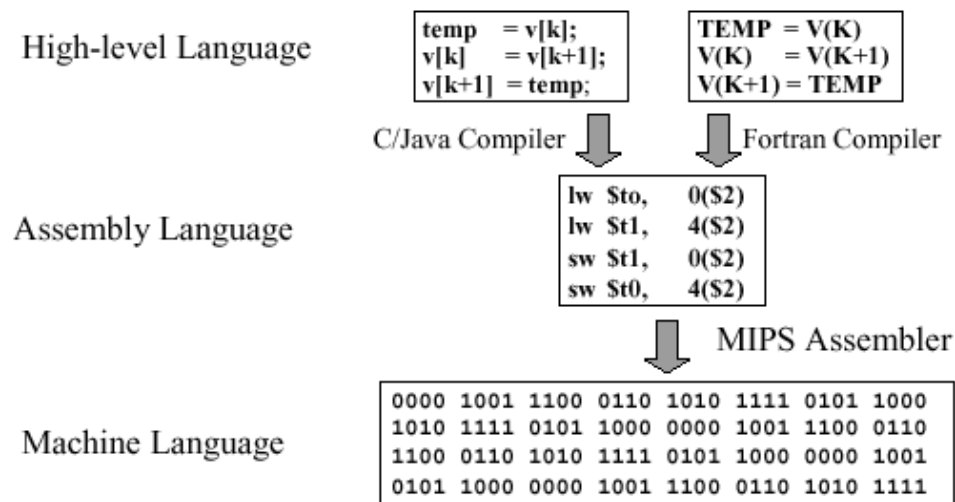
TED UNIVERSITY, COURSE SYLLABUS			
Faculty	Engineering	Department	Computer Engineering
Course Code & Number	CMPE 252	Course Title	C Programming
Type of Course	<input checked="" type="checkbox"/> Compulsory <input type="checkbox"/> Elective	Semester	<input type="checkbox"/> Fall <input checked="" type="checkbox"/> Spring <input type="checkbox"/> Summer
Level of Course	BSc	Year of Study	Sophomore
Course Credit Hours	(2+0+2) 3	Number of ECTS Credits	6
Pre-requisite	N/A	Co-requisite	N/A
Mode of Delivery	<input type="checkbox"/> Face-to-face <input checked="" type="checkbox"/> Distance learning	Language of Instruction	<input checked="" type="checkbox"/> English <input type="checkbox"/> Turkish
Course Coordinator	Asst. Prof. Aslı Gençtav	Course Lecturers	Asst. Prof. Aslı Gençtav Assoc. Prof. <u>Gökçe Nur Yılmaz</u> Assoc. Prof. <u>Kasım Öztoprak</u>
Required Reading	Problem Solving and Program Design in C, Jeri R. Hanly & Elliot B. Koffman, Pearson 8. Edition, Global Edition	Course Assistant(s)	<u>Bedrettin Cetinkaya</u> <u>Mehmet Bahadır Askın</u> <u>Nigar Ercan</u>

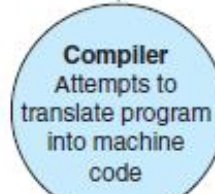
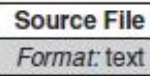
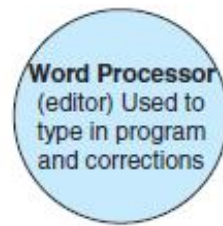
# Outline

- Syllabus
- **Overview of Computers and Programming, Chapter 1**
  - Importance of learning multiple programming languages
- Overview of C, Chapter 2

# Computer Languages

- machine language
  - binary number codes understood by a specific CPU
- assembly language
  - mnemonic codes that correspond to machine language instructions
- high-level language
  - machine-independent programming language that combines algebraic expressions and English symbols

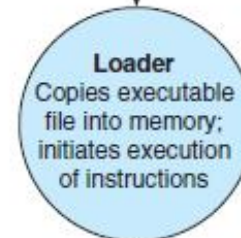
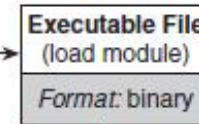
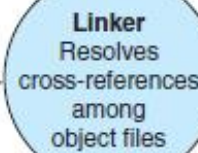
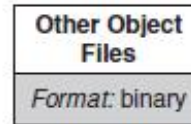
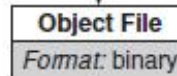




Unsuccessful



Successful



**source file:** file containing a program written in a high-level language; the input for a compiler

**compiler:** software that translates a high-level language program into machine language → **object file**

**syntax:** grammar rules of a programming language

**object file:** file of machine language instructions that is the output of a compiler

**linker:** software that combines object files and resolves cross-references to create an **executable** machine language program → **executable file**

Figure 1.11 Entering, Translating, and Running a High-Level Language Program

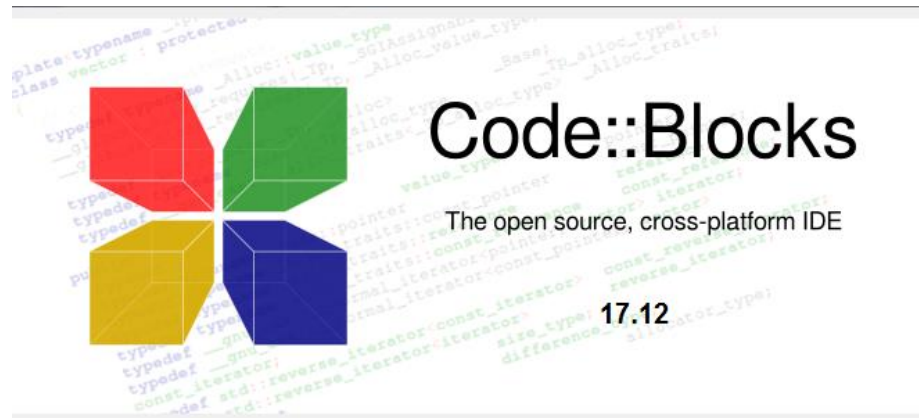
# Computer Languages

- integrated development environment (IDE)
  - software package combining a word processor, compiler, linker, loader, and tools for finding errors



# IDE WE USE

- We will use CodeBlocks



- [https://www.ntu.edu.sg/home/ehchua/programming/howto/CodeBlocks\\_HowTo.html](https://www.ntu.edu.sg/home/ehchua/programming/howto/CodeBlocks_HowTo.html)

# Some Basics


- If your OS is Windows, download the installer with GCC Compiler, e.g., `codeblocks-17.12mingw-setup.exe` which includes **MinGW**'s **GNU GCC** compiler and **GNU GDB** debugger.

What are they?



```
graph BT; A[What are they?] --> B[MinGW]; A --> C[GNU GCC]; A --> D[GNU GDB];
```

# GCC - GDB

- **GCC:** The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. Originally named the GNU C Compiler, when it only handled the C programming language, GCC 1.0 was released in 1987. It was extended to compile C++ in December of that year. Front ends were later developed for Objective-C, Objective-C++, Fortran, Java, Ada, and Go among others. [5]
  - **GDB:** GNU Project Debugger is a portable debugger that runs on many Unix-like systems and works for many programming languages, including Ada, C, C++, Objective-C, Free Pascal, Fortran, Go, Java and partially others.
- 

# GNU

- **GNU: GNU's Not Unix** (a recursive acronym)
- GNU Project: Announced on September 27, 1983 by Richard Stallman at MIT
- Purpose: Computer users should have a control in the use of their devices by collaboratively developing and providing software. They are free to run the software, copy it, share it, study it and modify it. GNU software guarantees these rights legally via its licence GNU GPL (general public licence).

# MinGW

- Minimalist GNU for Windows
- MinGW provides a complete Open Source programming tool set which is suitable for the development of native MS-Windows applications, and which do not depend on any 3rd-party C-Runtime **DLLs** or DLLs provided by Microsoft themselves, e.g. MSVCRT.DLL, the Microsoft C runtime library [6].



Okay then what is DLL?

**Dynamic-link library** is Microsoft's implementation of the shared library concept

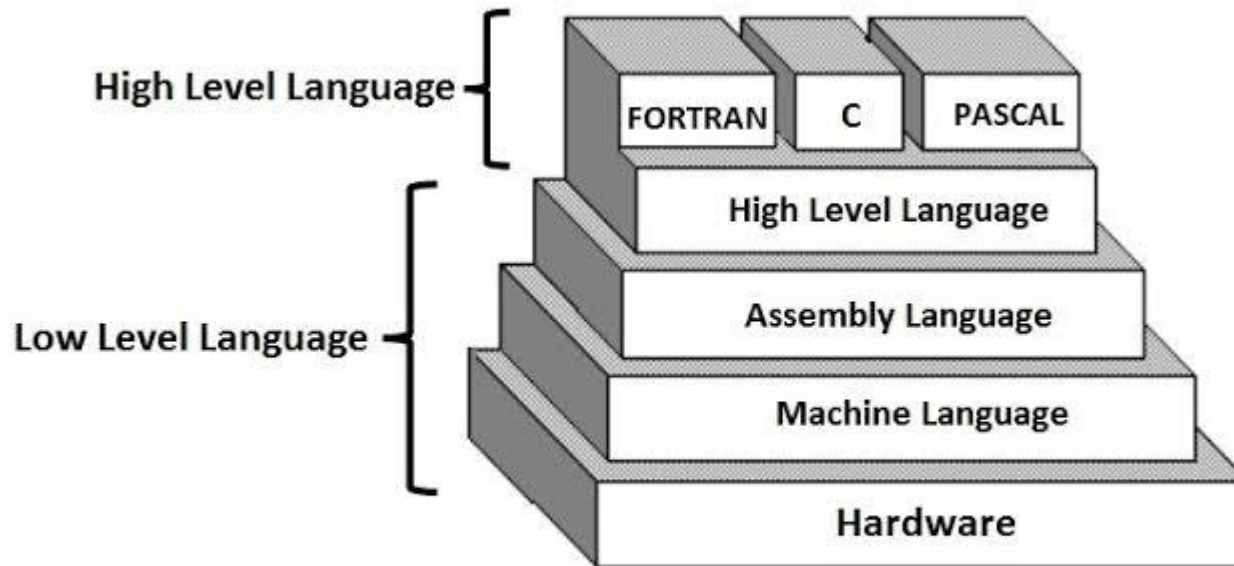
# Outline

- Syllabus
- Overview of Computers and Programming, Chapter 1
  - **Importance of learning multiple programming languages**
- Overview of C, Chapter 2

# Why learn Multiple Programming Languages?

- The initial question: Why are there multiple programming languages ?
- Scientific and technological developments are not performed from one center,
  - many emerge from different locations, different universities, teams/companies working parallel.
  - These works are performed due to different needs, to overcome different problems for different studies.

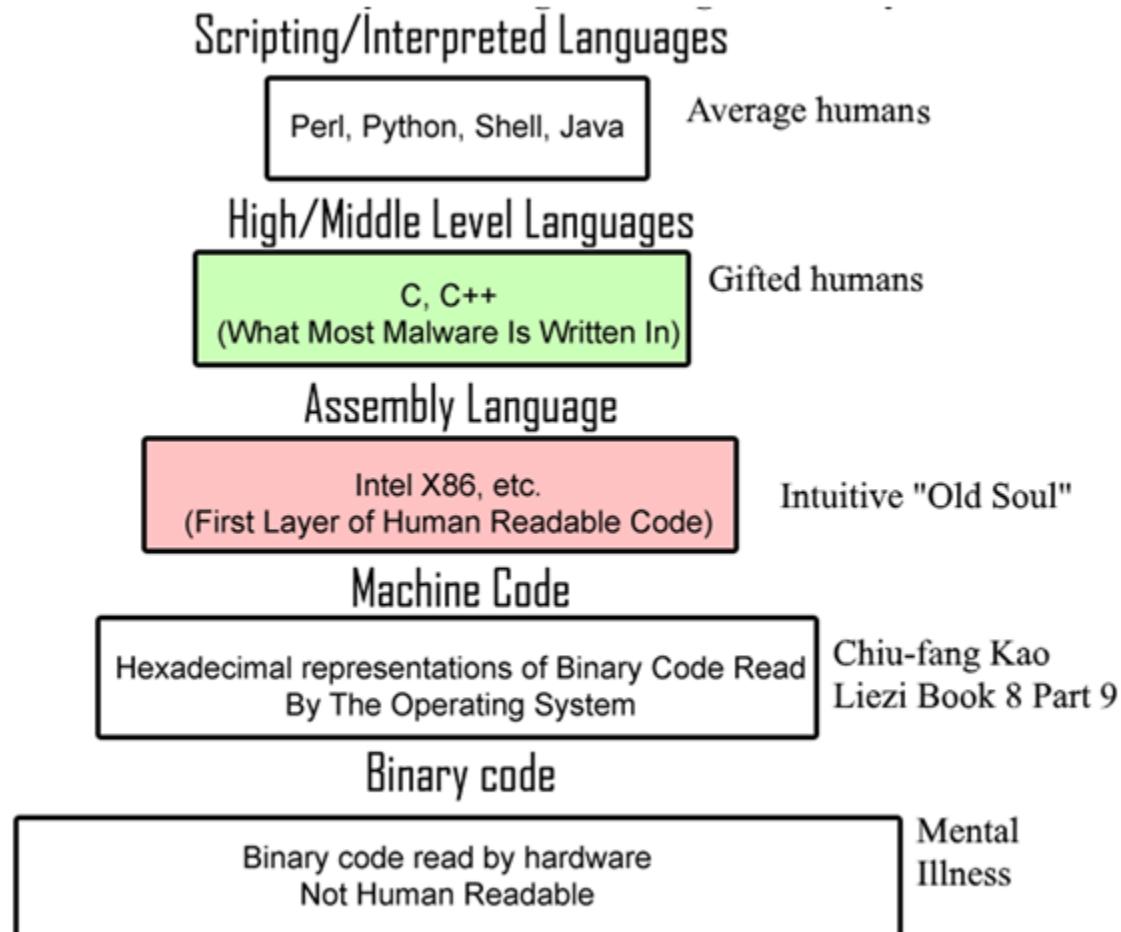
# Types of Languages



**Computer Language and its Types**



# More recent classification



# Different PLs better for different problems

- **Python:**
  - **Usage/Application:** Web and Internet Development, Scientific and Numeric applications, Desktop GUIs, Business applications. It is widely used in AI and Machine Learning space.
  - **Cons:**
    - Not ideal for Mobile Computing
    - Python's database access layer is bit underdeveloped and primitive.
- **Java:**
  - **Usage/Application:** Java mostly used for developing Android apps, web apps, Desktop GUIs and Big data.
  - **Cons.**
    - Memory management in Java is quite expensive
    - The absence of templates can limit you to create high-quality data structures.
- **C:**
  - **Usage/Application:** useful for operating systems, **embedded systems** or other programs where performance matters a lot (“high-level” interface would affect performance). Great control of memory management. Highly portable.
  - **Cons.**
    - Do not support OOP, namespaces, exception handling, runtime type checking. Abstraction and data hiding is hard. Small language.

# Top PLs in 2019/2020

*IEEE Spectrum's* sixth annual interactive ranking of the top programming languages

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	96.3
3	C	  	94.4
4	C++	  	87.5
5	R		81.5
6	JavaScript		79.4
7	C#	   	74.5
8	Matlab		70.6
9	Swift	 	69.1
10	Go	 	68.0

<https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019>

# Top PLs in 2020/2021

TIOBE Index for February 2021

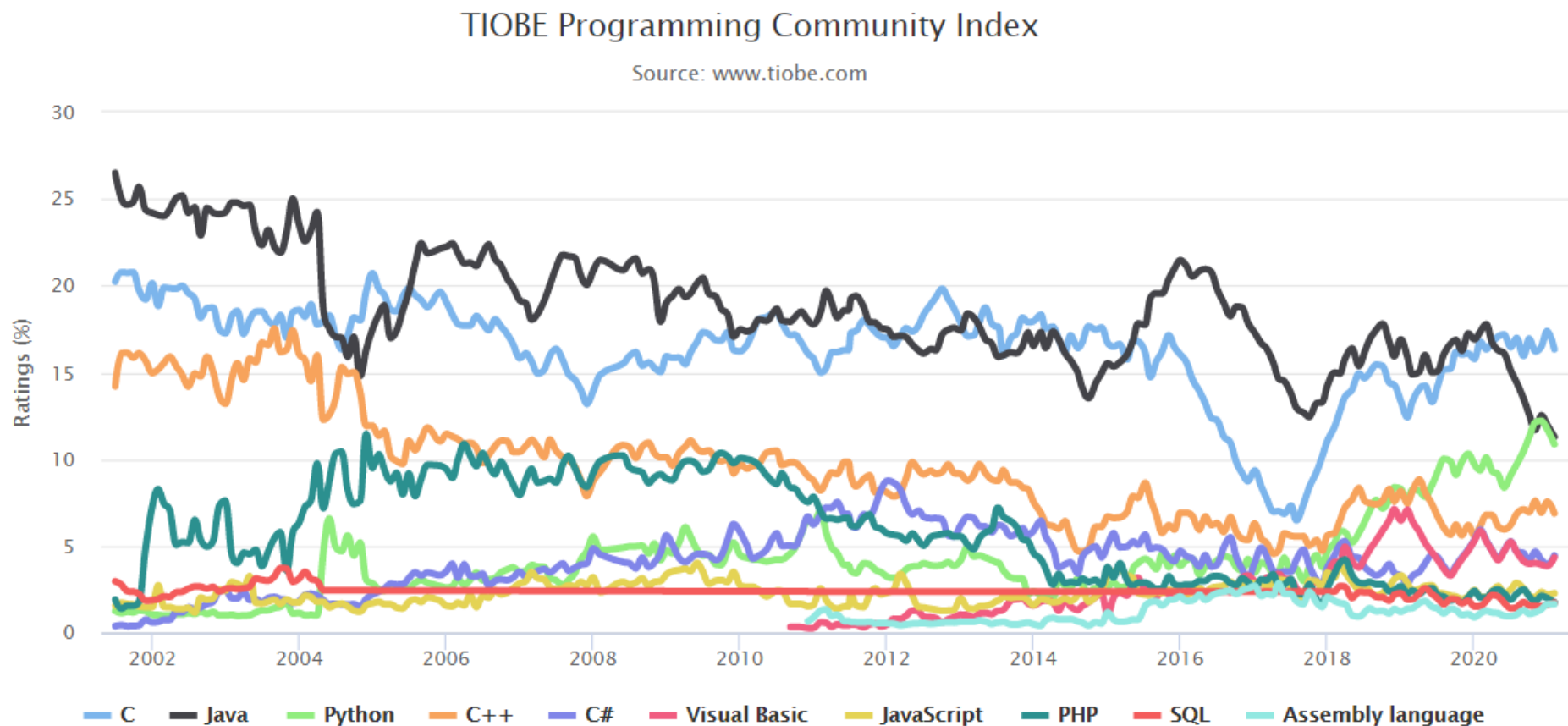
<https://www.tiobe.com/tiobe-index/>

Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	16.34%	-0.43%
2	1	▼	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	▲	Assembly language	1.65%	+0.54%
11	13	▲	R	1.56%	+0.55%
12	26	▲▲	Groovy	1.50%	+1.08%
13	11	▼	Go	1.28%	+0.15%
14	15	▲	Ruby	1.23%	+0.39%
15	10	▼▼	Swift	1.13%	-0.33%
16	16		MATLAB	1.06%	+0.27%
17	18	▲	Delphi/Object Pascal	1.02%	+0.27%

# Top PLs in 2020/2021

TIOBE Index for February 2021

<https://www.tiobe.com/tiobe-index/>



# Why Learning Multiple Programming Languages



More Knowledge



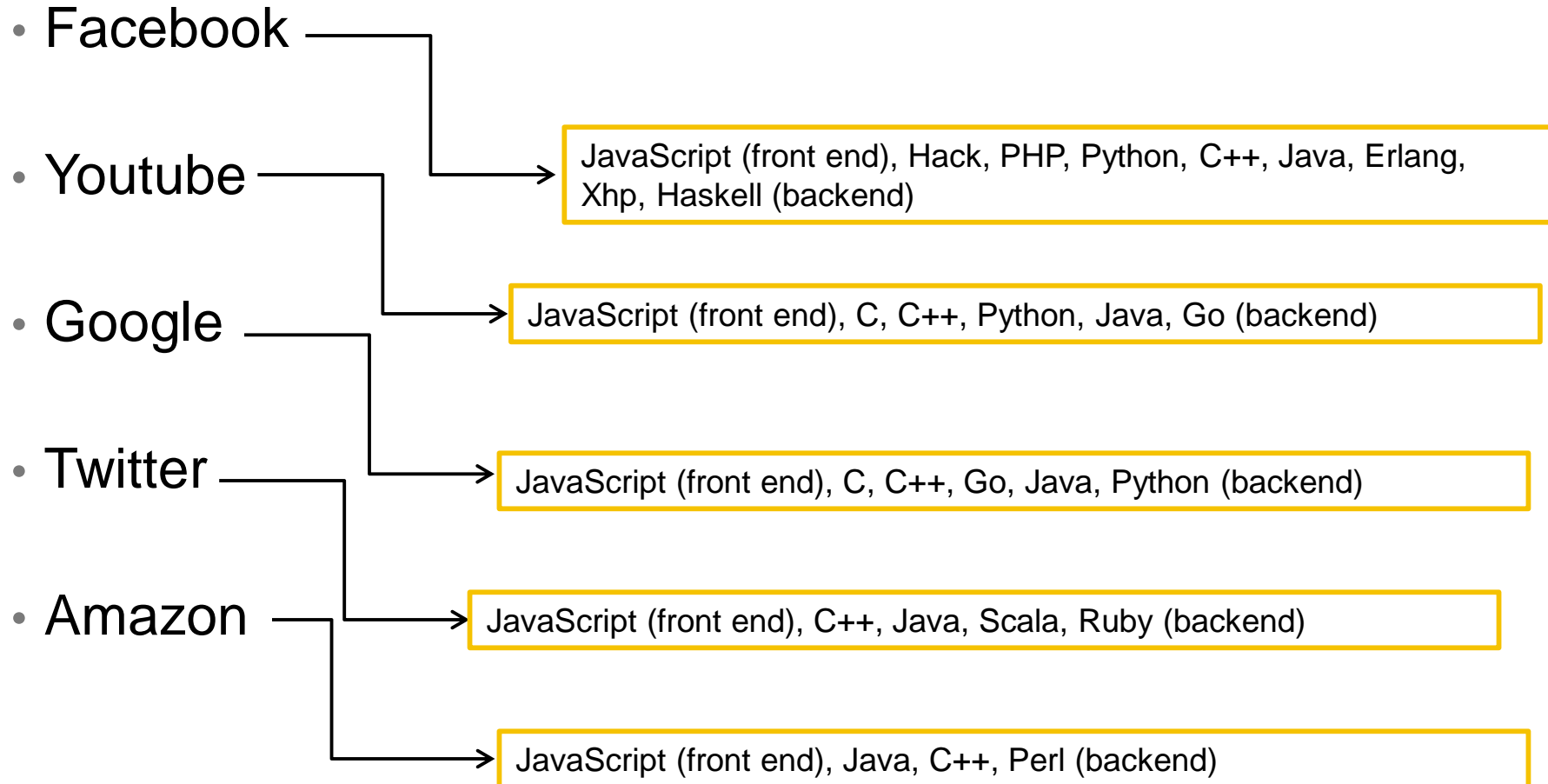
Better  
Opportunities



More  
Fun

- Similar to being bilingual
- Do not limit yourself while seeking a job
- Work on multiple projects if you wish, no more boredom → enjoy the work
- You never become outdated
- Adapt to industry trends

## [2,3] Languages used for...



# Outline

- Syllabus
- Overview of Computers and Programming, Chapter 1
  - Importance of learning multiple programming languages
- **C Basics and Why C**



# C

- Developed in 1972 by Dennis Ritchie at AT&T Bell Labs
- Designed as language to write the Unix operating system
- Resembles everyday English
- Very popular

# Uses of C

- C produces code that runs as fast as the code written in assembly language, some examples of the use of C:
  - Operating Systems (Unix, Linux, Windows kernel, MAC(OSX), iOS, Android)
  - Assemblers
  - Language Compilers C++, Lisp, Haskell
  - Language Interpreters MATLAB interpreter
  - Embedded Systems Almost all embedded systems
  - Text Editors Notepad++ written in C++
  - Network Drivers Oracle Database, MySQL, MS SQL Server, and PostgreSQL,
  - Databases
  - 3D Movies / Games OpenGL written in C
  - Utilities

# Using C

- Understanding Computer Architecture become easy
- Learning subjects becomes easier (by programming)
- Getting into IT companies is easy
- Understanding Electronics better
- Implementing Control Systems Better
  - Microcontroller and chip programming
- C is also the most widely used language for programming embedded processors
  - In 2005, 8.8. billion/ 9 bl. microprocessors were embedded.

# Using C

- High learning curve BUT:
- Valuable knowledge of compiling, optimizations, etc.
- Gain advanced programming skills
- Once you learn C, easy to step to other languages

# Basics

- C is a procedural language
  - A procedure is a sequence of instructions
- Task is solved in procedures
- Modular programming applies
- There are variables, data structures, functions
- Compared to OOP:
  - OOP breaks down the programming task into objects (methods+attributes)

# Java vs C

- No pointers in Java
  - Access to memory is easy in C
  - Performance is better in C if you write your code well
  - In C, near-to-machine level optimizations are possible
    - In Java, code interacts indirectly with the system using JVM which itself is another program running, which means extra overhead
- Java has garbage collector, so easier to handle memory leaks
  - C makes you understand the programming better since you will do many things by yourself

# C vs Java

- C is compiled faster (source code to binary file by a native compiler). Java is compiled slower (source code to intermediate byte code and interpreted to binary code)
- Functions, conditional statements, loops, arrays are similar
- Syntax is pretty close to each other

```
public class HelloWorld {  
    public static void main(String[] args){  
        System.out.println("Hello, World");  
    }  
}
```

```
#include <stdio.h>  
int main() {  
    printf("Hello, World!");  
    return 0;  
}
```

# OVERVIEW OF C

## CHAPTER 2

*Problem Solving & Program Design in C*

---

*Eighth Edition*

*Global Edition*

*Jeri R. Hanly & Elliot B. Koffman*

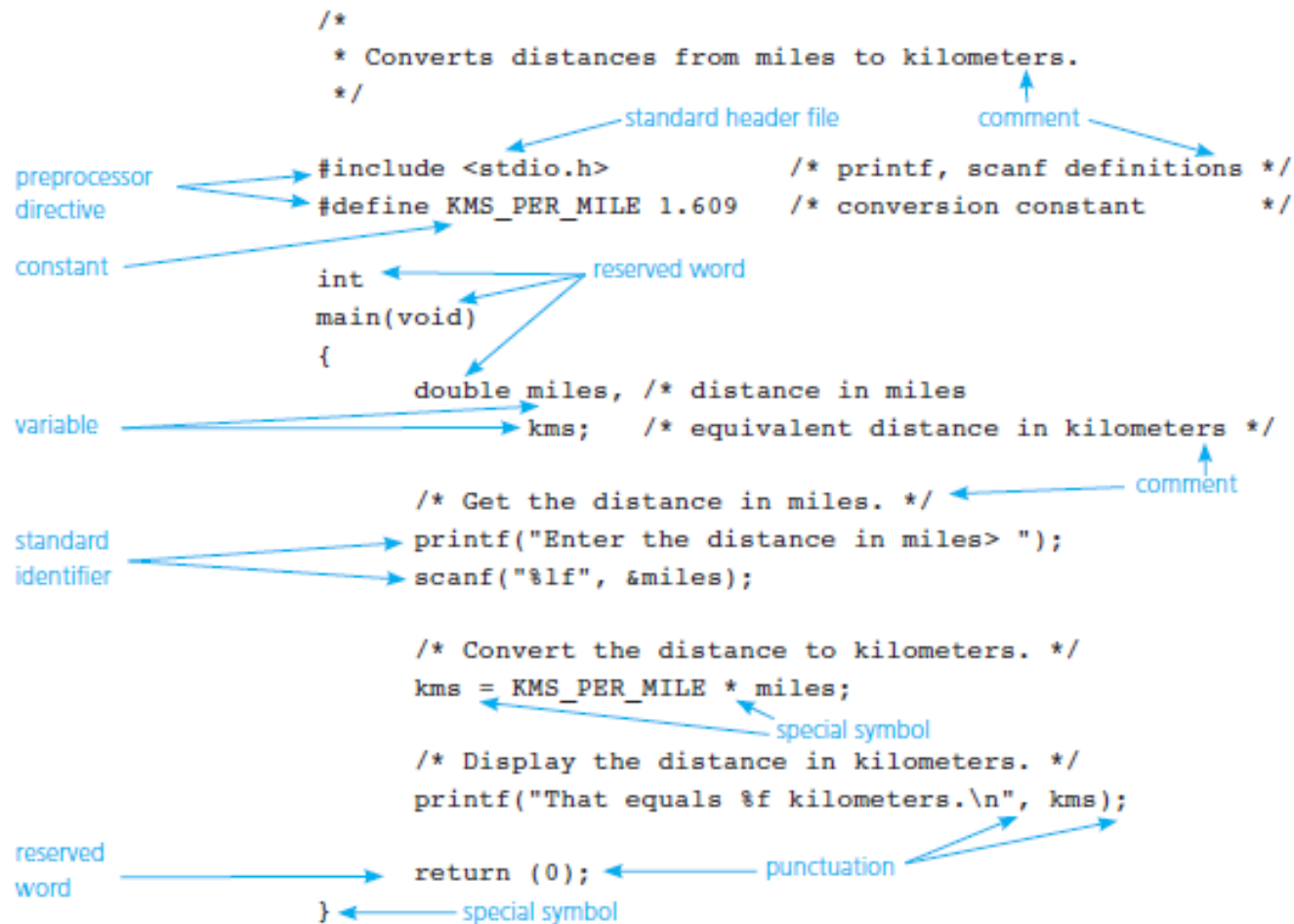


## Figure 2.8

### General Form of a C Program

```
preprocessor directives  
main function heading  
{  
    declarations  
    executable statements  
}
```

Figure 2.1 C Language Elements in  
Miles-to-Kilometers Conversion Program



# Program Style

- Use spaces consistently and carefully.
  - One is required between consecutive words in a program.
  - Improves readability.
- Use comments to document your program.
  - Also enhances readability.

# Language Elements

- comment
  - text beginning with `/*` and ending with `*/` that provides supplementary information but is ignored by the preprocessor and compiler

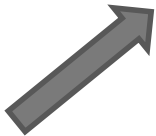
`/* Get the distance in miles */`

`// this is also a comment for 1 line only`

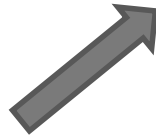
# Language Elements

- preprocessor
  - a system program that modifies a C program prior to its compilation
- preprocessor directive
  - a C program line beginning with # that provides an instruction to the preprocessor

#include <stdio.h>




#define KMS\_PER\_MILE 1.609



# Language Elements

- library
  - a collection of useful functions and symbols that may be accessed by a program
  - each library has a standard header file whose name ends with the symbols “.h”



```
#include <stdio.h> /*includes printf, scanf definitions*/  
#include <math.h> /*includes math definitions*/  
#include <string.h> /*includes string (text) manipulation functions*/  
... many more.
```

```
SYNTAX:      #include <standard header file>
```

***include*** directive gives a program access to a library

# Language Elements

- constant macro
  - a name that is replaced by a particular constant value before the program is sent to the compiler

`#define KMS_PER_MILE 1.609`

*constant*

*constant macro*

`kms = KMS_PER_MILE * miles;`

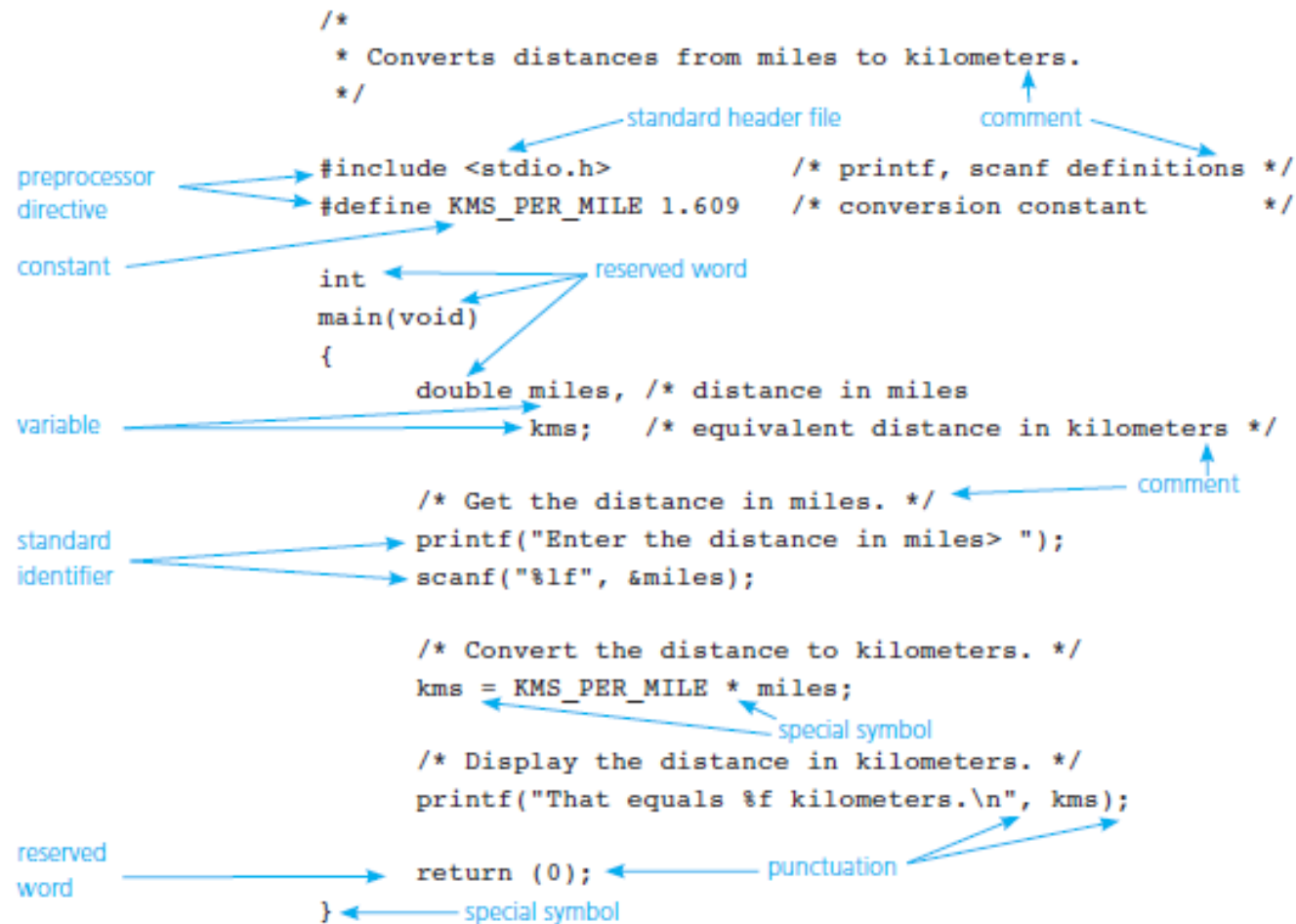
## `#define` Directive for Creating Constant Macros

SYNTAX: `#define NAME value`

EXAMPLES: `#define MILES_PER_KM 0.62137`  
`#define PI 3.141593`  
`#define MAX_LENGTH 100`

INTERPRETATION: The C preprocessor is notified that it is to replace each use of the identifier *NAME* by *value*. C program statements cannot change the value associated with *NAME*.

Figure 2.1 C Language Elements in  
Miles-to-Kilometers Conversion Program





# Function main

- Every C program has a main function.

int main (void)

OR

int  
main (void)

- These line(s) mark the beginning of the main function where program execution begins.

## main Function Definition

```
SYNTAX:  int  
         main(void)  
         {  
           function body  
         }
```

curly braces are used, to show to the compiler where a function starts and ends.

# Function main

- declarations
  - the part of a program that tells the compiler the names of memory cells in a program
- executable statements
  - program lines that are converted to machine language instructions and executed by the computer

```
int main(void)
{
    double miles; /* distance in miles */
    double kms; /* equivalent distance in kilometers */
```

declarations

```
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);
```

executable statements

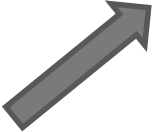
# Language Elements


- reserved word
  - a word that has a special meaning in C
  - identifiers from standard library and names for memory cells
  - appear in lowercase
  - cannot be used for other purposes

auto	default	float	register	switch
break	do	for	return	typedef
case	double	goto	short	union
char	else	if	signed	unsigned
const	enum	int	sizeof	void
continue	extern	long	static	volatile
			struct	while

# Language Elements

- standard identifier
  - a word having special meaning but one that a programmer may redefine
    - Correspond to standard functions already implemented for a purpose

 `printf("Enter the distance in miles> ");` from **stdio.h**

 `sin(PI);` from **math.h**

In C, standard identifiers (standard functions) can be redefined, but *redefinition is not recommended*.

# Variable Declarations

- variable
  - a name associated with a memory cell whose value can change
- variable declarations
  - statements that communicate to the compiler the names of variables in the program and the kind of information stored in each variable

```
double miles; /* distance in miles */  
double kms; /* equivalent distance in kilometers */
```

# Variable Declarations

- C requires you to declare every variable used in a program.
- A variable declaration begins with an identifier that tells the C compiler the type of data stored in a particular variable.

`int hours;`

`double miles;`

## Syntax Display for Declarations

SYNTAX:     `int variable_list;`  
              `double variable_list;`  
              `char variable_list;`

EXAMPLES:   `int count,`  
              `large;`  
              `double x, y, z;`  
              `char first_initial;`  
              `char ans;`

INTERPRETATION: A memory cell is allocated for each name in the *variable\_list*. The type of data (`double`, `int`, `char`) to be stored in each variable is specified at the beginning of the statement. One statement may extend over multiple lines. A single data type can appear in more than one variable declaration, so the following two declaration sections are equally acceptable ways of declaring the variables `rate`, `time`, and `age`.

<code>double rate, time;</code>		<code>double     rate;</code>
<code>int age;</code>		<code>int        age;</code>
		<code>double     time;</code>

# Data Types

- **int**
  - a whole number
  - 435
- **double**
  - a real number with an integral part and a fractional part separated by a decimal point
  - 3.14159
- **char**
  - an individual character value
  - enclosed in single quotes
  - 'A', 'z', '2', '9', '\*', '!'



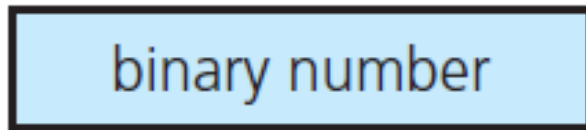
**TABLE 2.4** Type double Constants (real numbers)

Valid double Constants	Invalid double Constants
3.14159	150 (no decimal point)
0.005	.12345e (missing exponent)
12345.0	15e-0.3 (0.3 is invalid exponent)
15.0e-04 (value is 0.0015)	
2.345e2 (value is 234.5)	12.5e.3 (.3 is invalid exponent)
1.15e-3 (value is 0.00115)	34,500.99 (comma is not allowed)
12e+5 (value is 1200000.0)	

# Figure 2.2

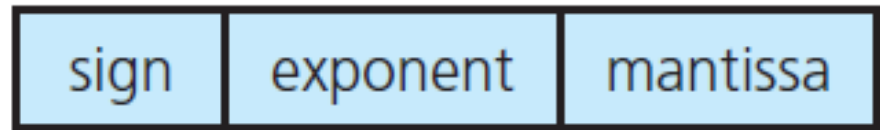
## Internal Format of Type int and Type double

type `int` format



32-bits

type `double` format



1 bit

11 bits

52 bits

64-bits

**double:** The storage area occupied by the number is divided into three sections:  
**sign** is 1 bit (0 for positive numbers, 1 for negative numbers),  
**mantissa** is a binary fraction  
**exponent** is an integer.  
mantissa and exponent are chosen so that  $real\ number = mantissa \times 2^{exponent}$

# HOA 1, Q1-2

# User-defined identifiers

- These name memory cells that hold data and program results and to name operations that we define.
- Naming rules:
  1. An identifier must consist only of letters, digits and underscores.
  2. An identifier cannot begin with a digit.
  3. A C reserved word cannot be used as an identifier.
  4. An identifier defined in a C standard library should not be redefined.
  5. C is case sensitive. rate, Rate and RATE are different identifiers.

# User-defined identifiers

- Quick Check: Which identifiers are valid?
  - letter\_1 ✓
  - 1Letter ✗
  - CENT\_PER\_INCH ✓
  - variable ✓
  - TWO\*FOUR ✗
- What happens when we define below two identifiers?
  - per\_capita\_meat\_consumption\_in\_1980 ✓
  - per\_capita\_meat\_consumption\_in\_1995 ✓
  - Both are syntactically correct and there is no rule about length, BUT
    - some ANSI C compilers consider only first 31 characters, therefore, would consider these two identifiers identical

Figure 2.1 C Language Elements in Miles-to-Kilometers Conversion Program

