CS 1037
Computer Science Fundamentals II

Part One: Basic C

INTRODUCTION TO PROGRAMMING

Program – A set of instructions that a computer uses to do something.

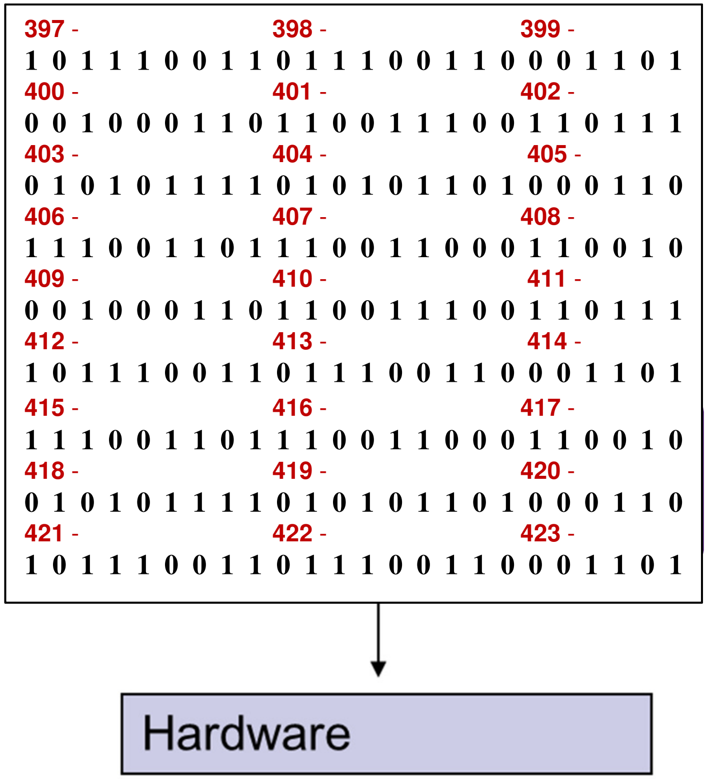
Programming / Develop – The act of creating or changing a program

Programmer / Developer – A person who makes a program

Run / Execute – The act of using a program

- Every program was created by someone
- Computers use special languages
- Programmers use special languages to create or change a program

Machine Language

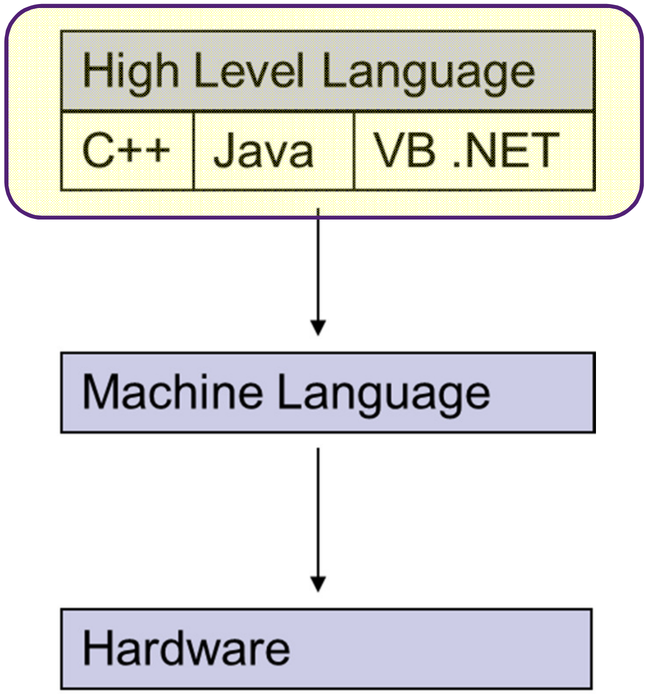


```
397 -             398 -             399 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1
400 -             401 -             402 -
0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1
403 -             404 -             405 -
0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0
406 -             407 -             408 -
1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0
409 -             410 -             411 -
0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1
412 -             413 -             414 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1
415 -             416 -             417 -
1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0
418 -             419 -             420 -
0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0
421 -             422 -             423 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1
```

Hardware

- Machine language – A language understood by computers
- When programs are run, machine language is used
- Machine languages are almost impossible for humans to understand
- Every operating system (OS) has its own machine language
  - Windows
  - Linux
  - Macintosh

High Level Languages



High Level Language

C++ | Java | VB .NET

Machine Language

Hardware

- High Level Language - A programming language that is understandable by people
- This enables a programmer to write programs
- High level languages must be translated into machine language before running on a computer

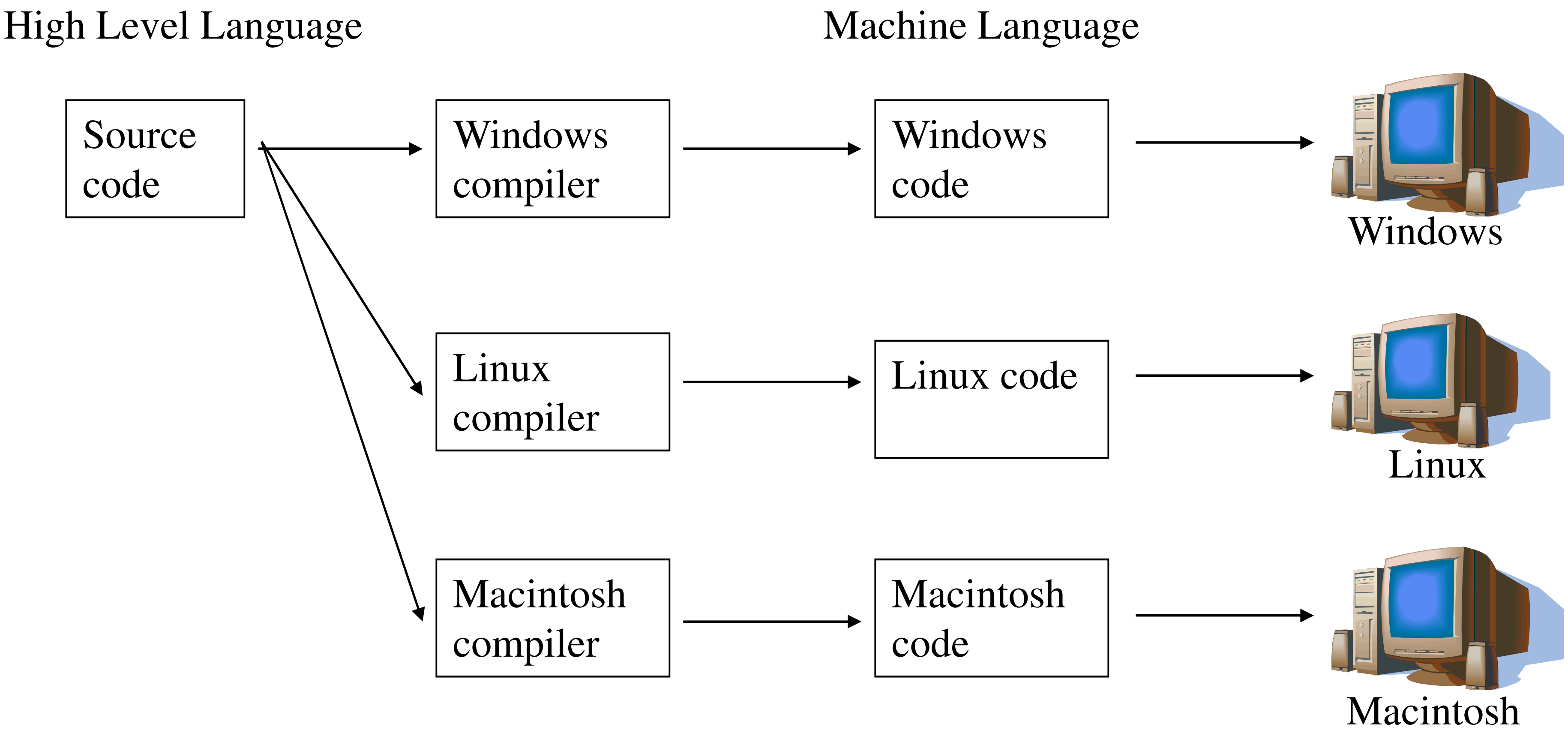
Compilers and Interpreters

- There are two main ways to change programs written in a high level language to machine language:
  1. Use a compiler
  2. Use an interpreter

- **Source Code** – Code written in a programming language by a developer.

Compilers

- **Compiler** – A program that transforms code from one format to another



High Level Language

Machine Language

Source code

Windows compiler

Windows code

Windows

Linux compiler

Linux code

Linux

Macintosh compiler

Macintosh code

Macintosh

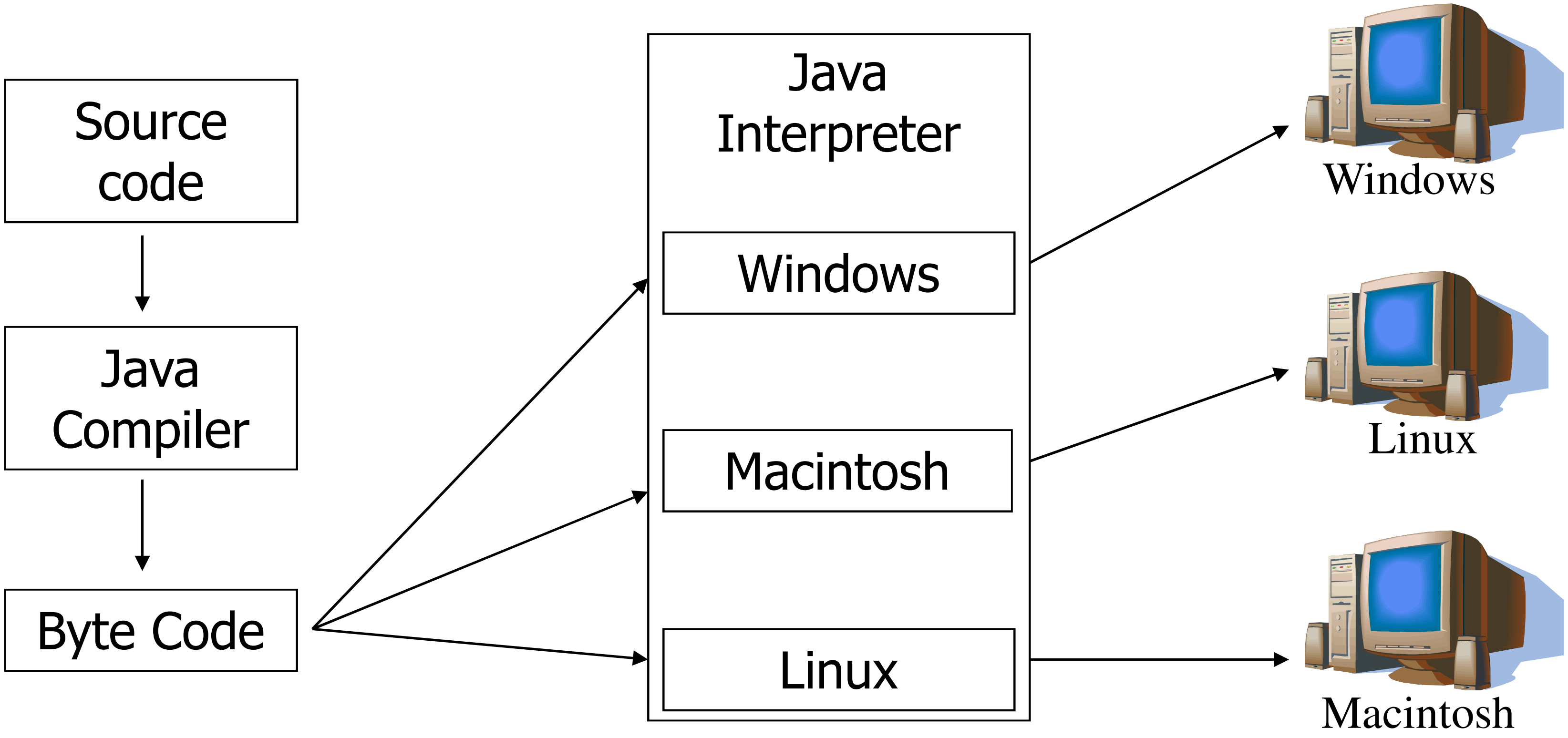
Compilers vs. Interpreters

- Advantages
  - Programs run faster

- Disadvantages
  - Platform dependent - Programs only work on a specific operating system.
    - Windows
    - Linux
    - Macintosh
  - Compiling a large program may take a long time

Interpreters

- **Interpreter** – A program that translates and executes code
- Usually, interpreters translate and execute source code.



Interpreters vs. Compilers

- Advantages
  - Platform Independent.
  - Don’t need to compile anything.
- Disadvantages
  - The interpreter program must be installed on the computer that the program will run on
  - Slower execution
  - You should only use services available on all platforms.
    - Example: Windows has a cool sound library, but you can’t use it because it won’t run on Macintosh

C versus C++

C is a system programming language
whereas **C++** is a general-purpose programming language
commonly used in embedded systems. **C** is procedural

C does not support classes and objects like **C++** does
(although, despite being object-oriented, **C++** can be procedural like **C**,
making it a bit more hybrid).

Generally, you'd opt to use **C** over **C++** if you didn't want the extra overhead of **C++**

C is good for embedded devices, networking, gaming and system level code
C++ is good for server-side applications and device drivers

BUT !!!
The whole **C++** language is indeed a monstrosity designed by a committee
that resulted from years of piling up "requested features",

C is not an object orient programming language !

Students in the prerequisite for this course learned the Java programming language

Fact:

Java is an object-oriented programming language

Trade mark of Object-oriented programming language:

- Object-oriented programming languages provide a programming construct to associate data (variables) and program code (methods).

- The program code associated with the data has special access permission to the data
- Example:
  In Java, only the methods (code) in the same class as
  the variables (data) can access the private variables
  defined inside that class !!!

Fact:

C does not provide any mechanism to associate data (variables) and code (methods/functions)

JAVA CODE

```java
public class HelloWorld {

     public static void main(String[] args) {
          // Prints "Hello, World" to the terminal window.
          System.out.println("Hello, World");
     }
}
```

C CODE

```c
#include <stdio.h>
int main()
{
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

Things that are identical in Java and C:

Due to the fact that Java is derived (indirectly) from C,
and the fact that you already know Java,
you have seen many features of C already !!!

- **Data types**

  Similar data types in C and Java

  - int
  - short
  - long
  - float
  - double

- **Variable definition syntax**

  Syntax to define variables:

  - int x;
  - double y;

Things that are identical in Java and C:

- **Arithmetic operators**

  Arithmentic operators: (add, subtract, multiply, divide, (modulo))

  - Integer: + - * / %
  - Float: + - * /

- **Increment/decrement operators**

  Increment/decrement operators: (++, --)

  - Pre operators: ++x --x
  - Post operators: x++ x--

- **Assignment operators**

  Assignment operators

  - Integer Assignment operators: += -= *= /= %=
  - Float Assignment operators: += -= *= /=

Things that are identical in Java and C:

- **Comparison and Logical operators**

  Comaprison operators: (less than, less than or equal, and so on)

  - Comparison operators: < <= > >= == !=

  Logical operators: (And, Or, Not)

  - Logical operators: && || !

- **Statements**

  Syntax of all statements (assignment) are identical in C and Java

  - x = (a + b) * (c + d % e);

Things that are identical in Java and C:

- **Statements**

  Syntax of all statements (if, if-else, switch, while, for, do) are identical in C and Java

  -
    ```
    if ( a > b )
        max = a;
    else
        max = b;
    ```

  -
    ```
    while ( x < 10 )
        x++;
    ```

  -
    ```
    for ( i = 0; i < n; i++ )
        sum += i;
    ```

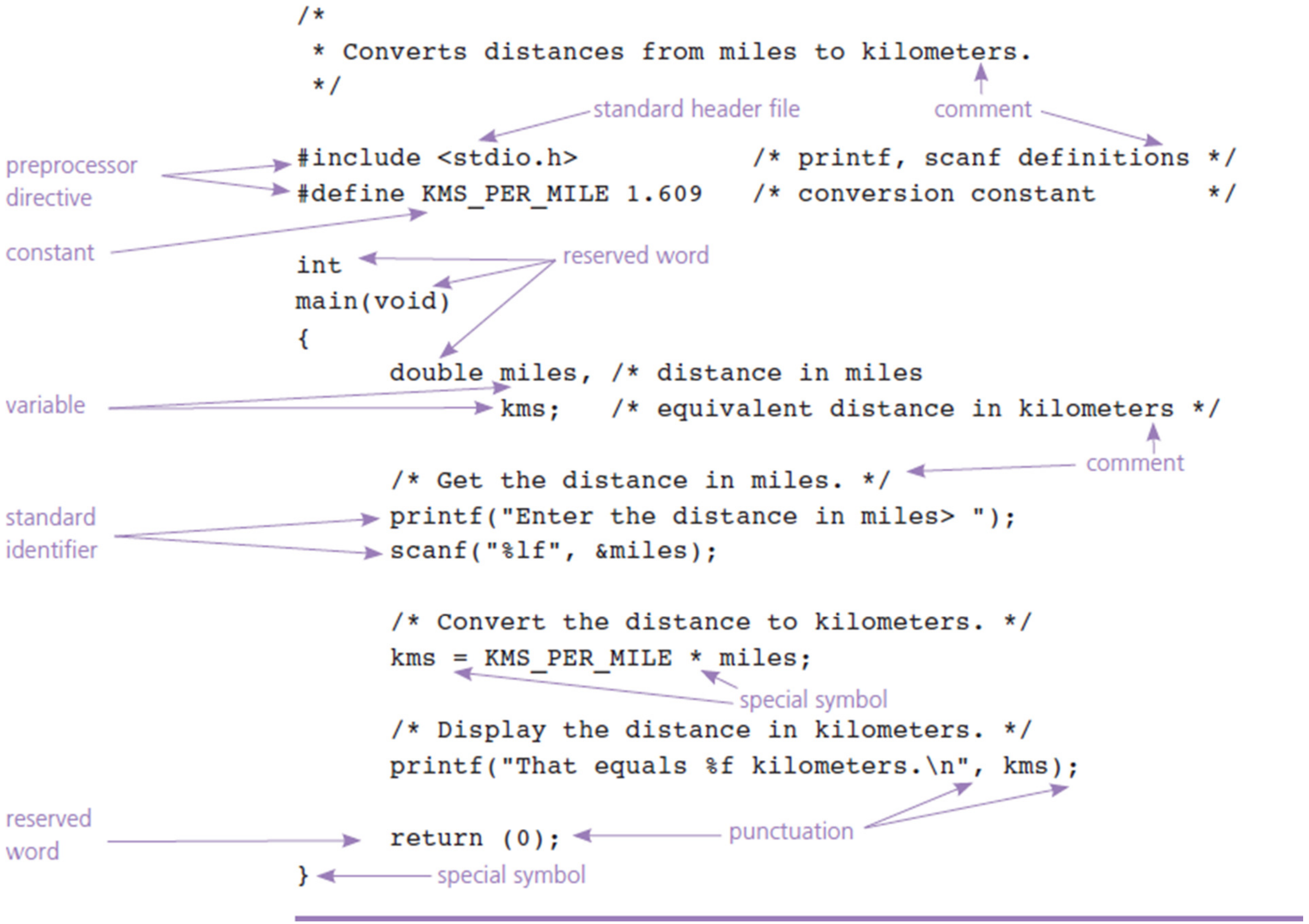
Structure of a C program:

A C program consists of a collection of:

- Data structures/types definitions
- (Global) variables
- Functions (with local variables and statements) stored in one or more files.

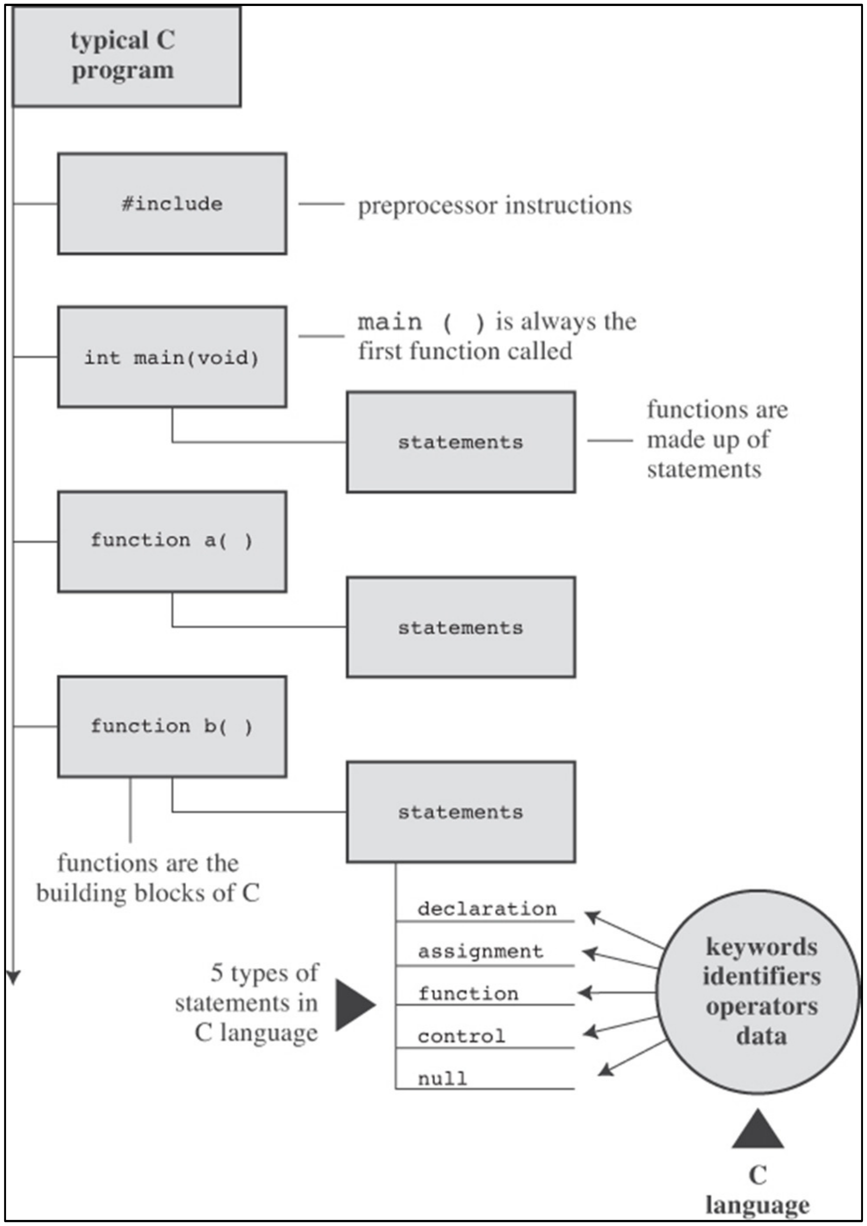
```
#include <stdio.h>

int f(float x)
{
  return (int) (x*x);
}

void main(int argc, char * argv[])
{
  int a;
  float b;

  a = 4;
  b = f(a);
  printf("a = %d, b = %f\n", a, b);
}
```

ANATOMY OF A C PROGRAM

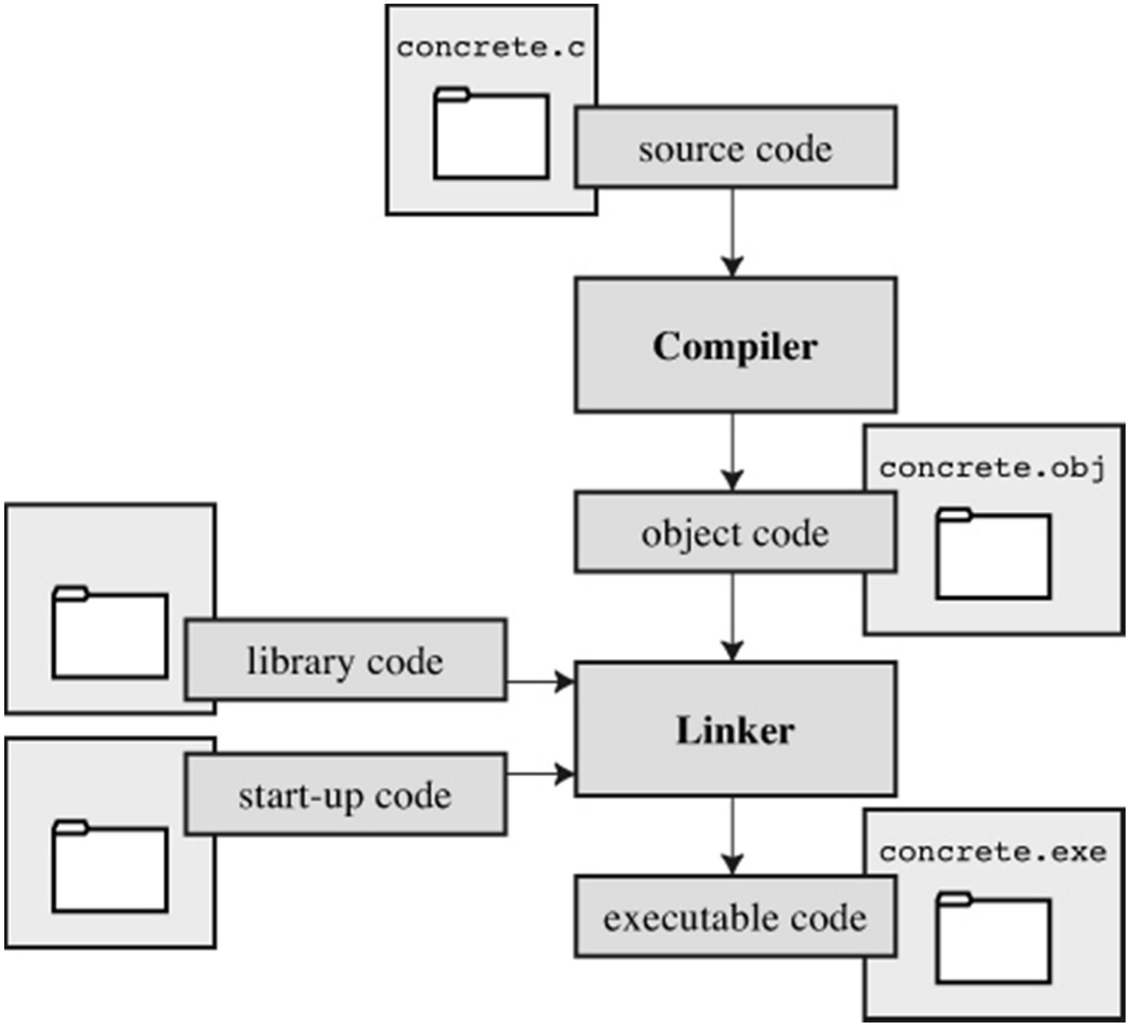


```
/*
 * Converts distances from miles to kilometers.
 */

#include <stdio.h>           /* printf, scanf definitions */
#define KMS_PER_MILE 1.609   /* conversion constant       */

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

      /* Get the distance in miles. */
      printf("Enter the distance in miles> ");
      scanf("%lf", &miles);

      /* Convert the distance to kilometers. */
      kms = KMS_PER_MILE * miles;

      /* Display the distance in kilometers. */
      printf("That equals %f kilometers.\n", kms);

      return (0);
}
```

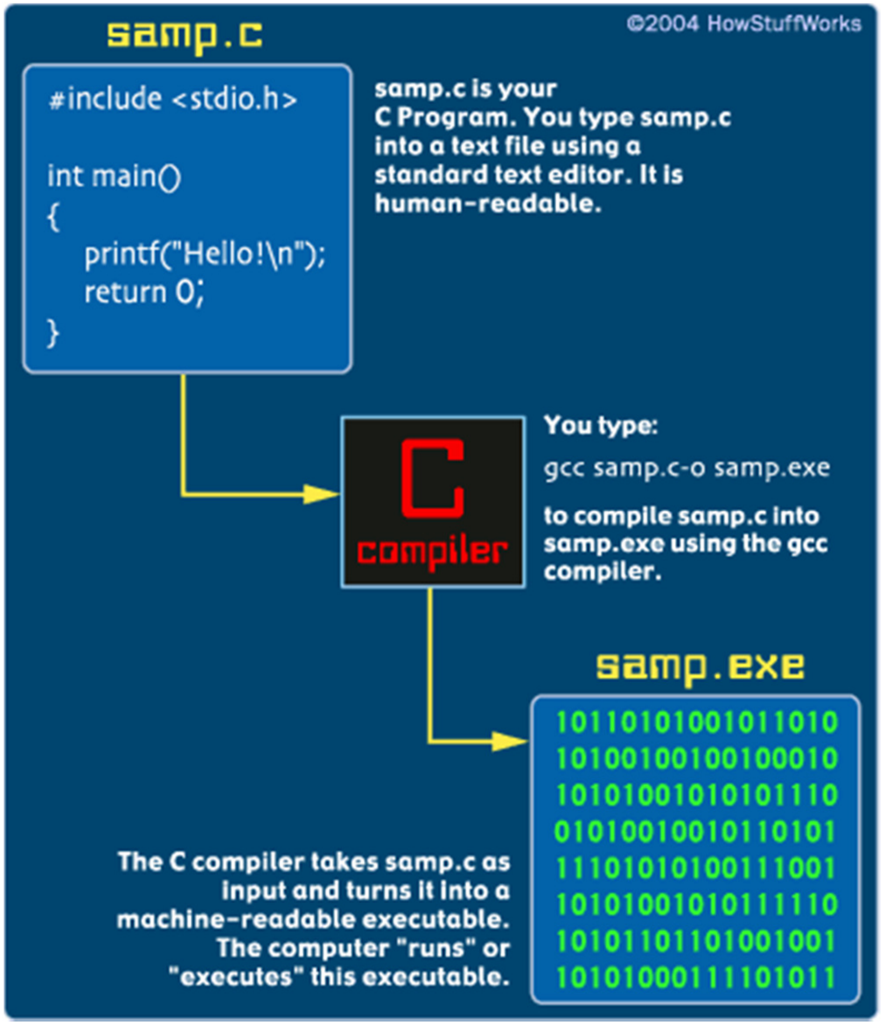
LAYERS of a C PROGRAM


typical C program
#include
preprocessor instructions
int main(void)
main ( ) is always the first function called
statements
functions are made up of statements
function a( )
statements
function b( )
statements
functions are the building blocks of C
5 types of statements in C language
declaration
assignment
function
control
null
keywords
identifiers
operators
data
C language

COMPILING a C PROGRAM



COMPILING a C PROGRAM



```
#include <stdio.h>

int main()
{
    printf("Hello!\n");
    return 0;
}
```

samp.c is your C Program. You type samp.c into a text file using a standard text editor. It is human-readable.

You type:

```
gcc samp.c-o samp.exe
```

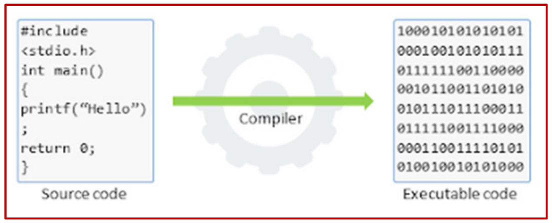
to compile samp.c into samp.exe using the gcc compiler.

```
1011010100101101O
1010010010010001O
1010100101010111O
0101001001011010 1
1110101010011100 1
1010100101011111O
1010110110100100 1
1010100011110101 1
```

The C compiler takes samp.c as input and turns it into a machine-readable executable. The computer "runs" or "executes" this executable.

COMPILING a C PROGRAM

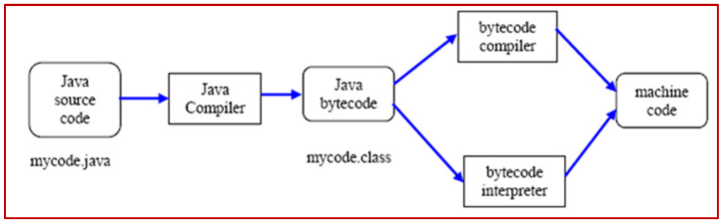
C : Execute the machine code directly by the computer:



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

```
100010101010101
000100101010111
011111100110000
001011001101010
010111011100011
011111001111000
000110011110101
010010010101000
```

JAVA: Execute the Java source code using a Java byte code interpreter (java)



Interpretation (executing code using an interpreter) is very inefficient.

Due to the fact that C program source is translated machine code,
C programs run multiple times (at least 10) faster than Java programs

COMPILING a C PROGRAM

Step 1: Write the source codes (.c)
and header files (.h).

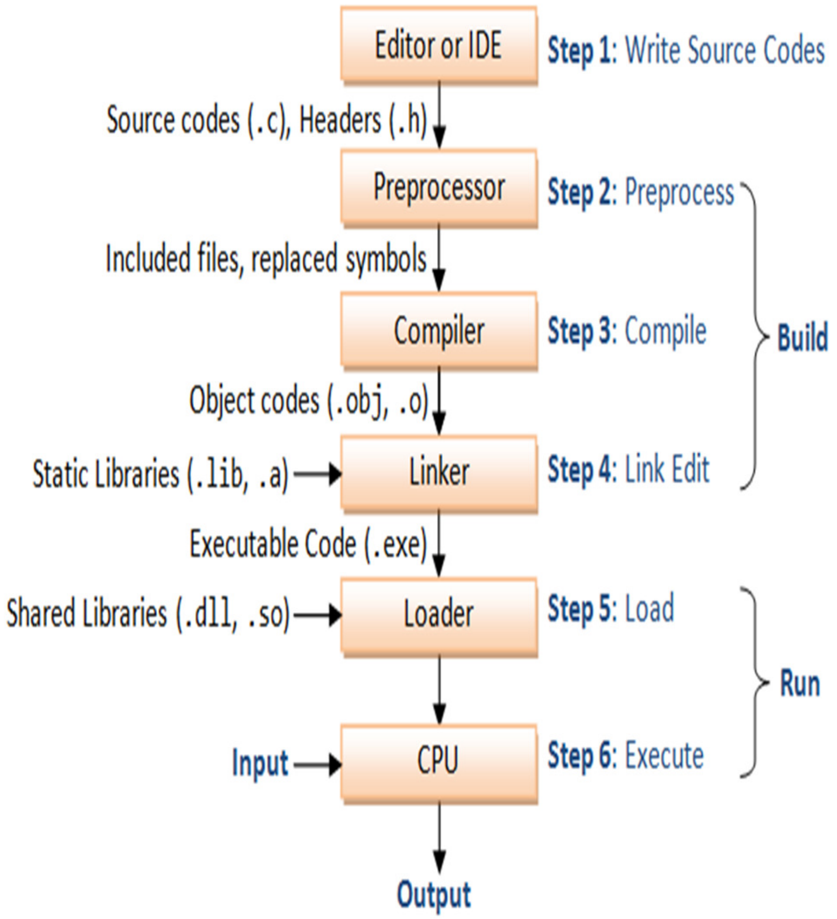
Step 2: Pre-process the source codes
according to the preprocessor
directives. The preprocessor
directives begin with a hash sign (#).

Step 3: Compile the pre-processed source
codes into object codes (.obj, .o).

Step 4: Link the compiled object codes with
other object codes and the
library object codes (.lib, .a)
to produce the executable code (.exe).

Step 5: Load the executable code into
computer memory.

Step 6: Run the executable code.



COMPILING a C PROGRAM

Step 2: Pre-process the source codes according to the preprocessor directives.

Before invoking the C compiler, the C programming language system will always invoke a **C pre-processor** to process the program source code.

Tasks performed by the C pre-processor:

Removes comments from the source code

```
        /* .......... */
  or:   // ........
```

Read in included file

```
      #include <stdio.h>
  or  #include "header.h"
```

Process macro (symbolic) definitions

```
  #define ...   ....
```

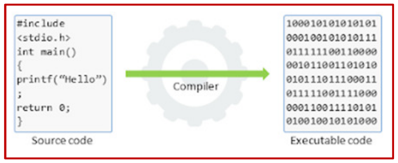
Other advanced conditionals:

```
  #ifdef  ...
    ....
  #endif


  #ifndef  ....
    ....
  #endif
```

COMPILING a C PROGRAM

Step 3: Compile the pre-processed source codes into object codes (.obj, .o).

A object file (.o) contains machine instructions in binary code
It's not for human consumption !



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

```
1000101010101010
0001001010101111
0111111001100000
0010110011010101
0101110111000111
0111110011110000
0001100111101011
0100100101010000
```

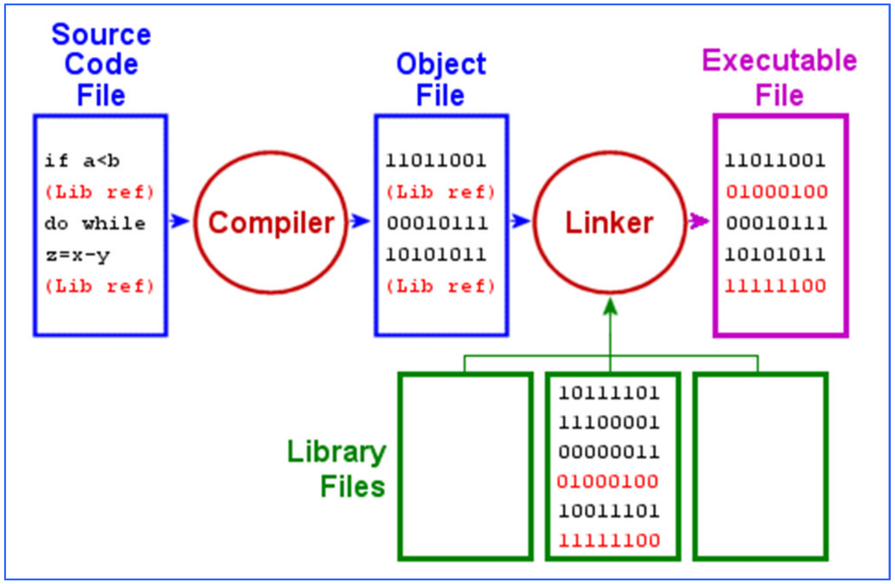
```
int f( int x )
  {
     return ( x*x );
  }
```

Compiled code:

```
0000000000000000 <f>:
  0: 0011010101101011        push   %rbp
  1: 1101010101000110        mov    %rsp,%rbp
  4: 1001011010100011        mov    %edi,-0x4(%rbp)
  7: 1011110101001000        mov    -0x4(%rbp),%eax      // get x in reg. eax
  a: 0011010110110011        imul   -0x4(%rbp),%eax      // (x*x)
  e: 0101010110001101        leaveq
  f: 1110111010101100        retq
```

COMPILING a C PROGRAM

Step 4: Link the compiled object codes with other object codes and the library object codes (.lib, .a) to produce the executable code (.exe).



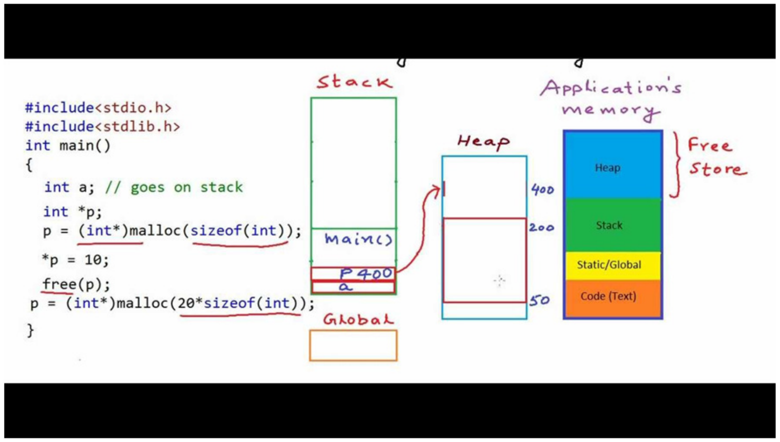
The linker will **delineate (assign) memory space** for every variable

So: the variables z, x and y will be stored somewhere.

Depending on where the variables are stored,
the linker will patch the relocation location with the allocated location

COMPILING a C PROGRAM

Step 5: Load the executable code into computer memory.
Step 6: Run the executable code.

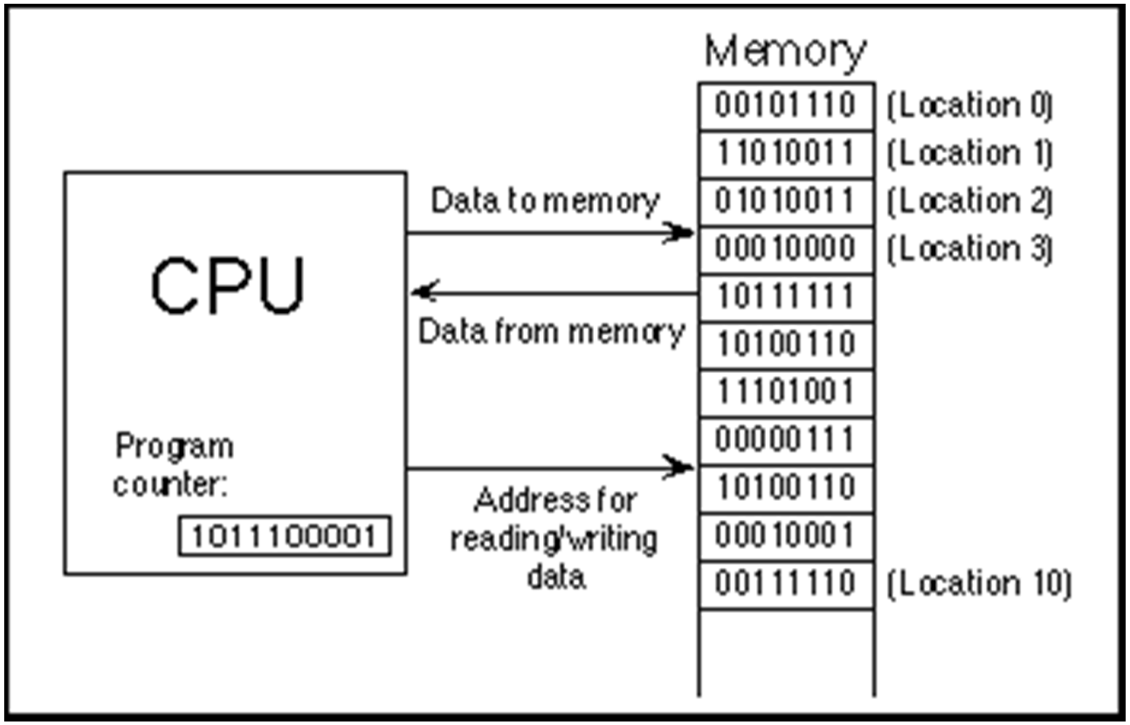


```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int a; // goes on stack
  int *p;
  p = (int*)malloc(sizeof(int));
  *p = 10;
  free(p);
 p = (int*)malloc(20*sizeof(int));
}
```

STACK: memory allocated in a stack frame when the function executes.
Released when function terminates

HEAP: dynamic allocated memory during execution. Memory survives function.
(which can be a cause of memory leaks if not garbage-collected.

RUNNING a C PROGRAM



Memory addressing (usage) is the basis of the C language