

# Structured Programming Language

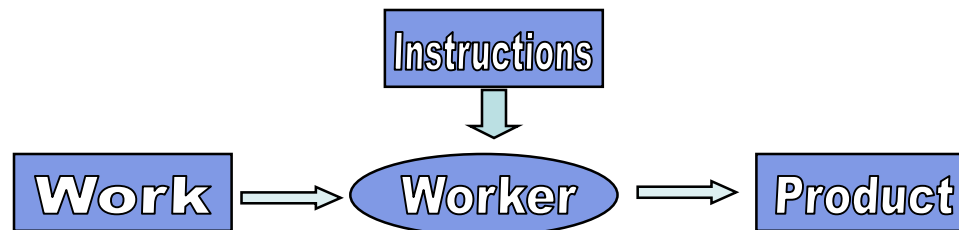
# Program

- A program is a **specific set** of operations to perform.
- A **computer program** is a **collection of *instructions*** that performs a **specific task** when executed by a computer.
- A computer program is usually written by a **computer programmer** in a **programming language**.

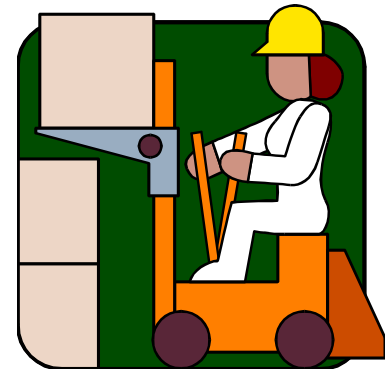
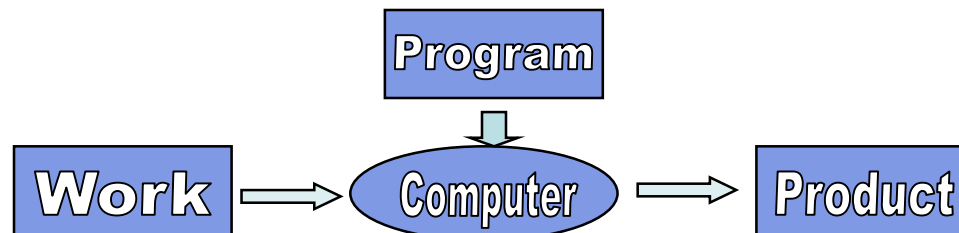
# Program Cont.

- A computer program performs a specific task, and may interact with the user and the computer hardware.

– Human work model:



– Computer work model:



# Language Types

- 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**

# Language Types, Cont.

## 3. High-level languages

- Codes similar to everyday English
- Use mathematical notations (translated via compilers)
- Example:

`grossPay = basePay + overTimePay`

# An example:

## Machine binary language

```
00001001001011100110011001101001011011000110010100001001001000
0101011100100110010100110001001011100110001100100010000010100:
0110001101101111011011010111000001101001011011000110010101100:
00110110010101100011011101000110100101101111011011100000100100
01110100001000100000101000001001001011100110000101101100011010
1010000010010010111001100111011011000110111101100010011000010:
0110111000001010000010010010111001110100011110010111000001100:
1001011011100010110000100011011001100111010101101110011000110:
00001001001011100111000001110010011011110110001100001001001100
1001011011100011101000001010000010010010000100100011010100000:
01010101010001010010001100100000001100000000101000001001011100
01010111001101110000001011000010110100110001001100100011100000
0000101000001001001000010010001101010000010100100100111101001:
00110010000000110001000010100000100101101101011011110111011001
0011000000001010000010010111001101110100001000000010010101101:
0110011100000010110100110010001100000101110100001010000010010:
0010110000100101011011110011000000001010000010010111001101110:
1100010110110010010101100110011100000010110100110010001101000:
00100000010110110010010101100110011100000010110100110010001100
0000000010100000100101101100011001000010000001011011001001010:
01011101001011000010010101101111001100010000101000001001011000
1111001100000010110000100101011011110011000100101100001001010:
0111010000100000001001010110111100110000001011000101101100100:
10000101110100001010000010010110110101101111011101100010000000
00001010000010010110001000100000001011100100110001001100001100
00000000101000101110010011000100110000110001001110100000101000
0000100101110010011001010111001101110100011011110111001001100:
0110011001010011000100111010000010100000100100101110011100110:
0110110101100001011010010110111000101100001011100100110001001100011001100110
110101100001011010010110111000001010000010010010111001101001011001000
001000100100011101000011010000110011101000100000001010001110100111001010100101001001000000011
0010001011100011100000101110001100010010001000001010
```

## Low-level assembly

```
main:
    !#PROLOGUE# 0
    save %sp,-128,%sp

    !#PROLOGUE# 1
    mov 1,%o0
    st %o0,[%fp-20]
    mov 2,%o0
    st %o0,[%fp-24]
    ld [%fp-20],%o0
    ld [%fp-24],%o1
    add %o0,%o1,%o0
    st %o0,[%fp-28]
    mov 0,%i0
    nop
```

## High-level

```
int main()
{
    int x, y, z;

    x = 1;
    y = 2;
    z = x+y;

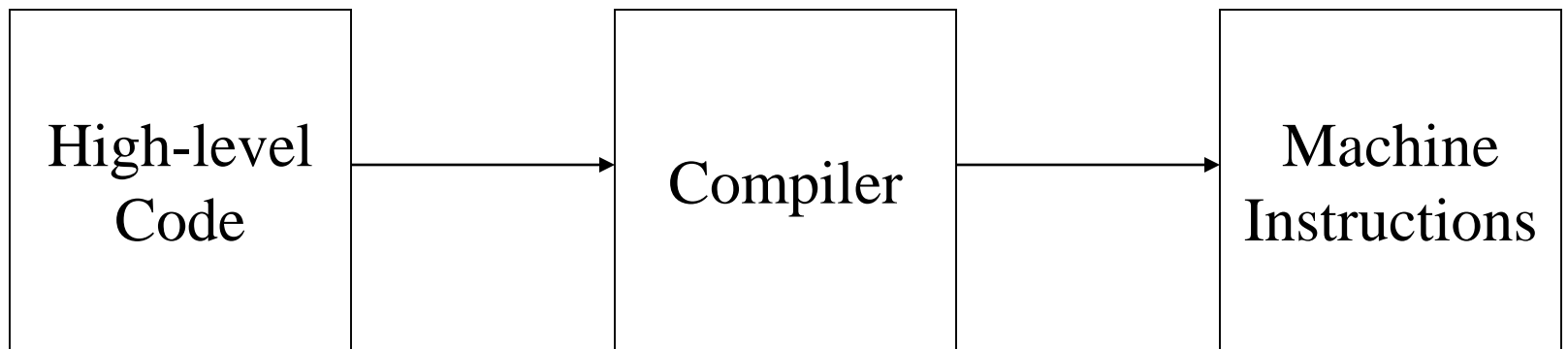
    return 0;
}
```

# High-level Languages

- “high-level” is a relative term
- C is a **relatively** low-level high-level language
- Pascal, Fortran, COBOL are typical high-level languages
- Java, Python, Perl, VB are examples of high-level high-level languages
- Application specific languages (Matlab, Javascript, VBScript, Scratch) are even higher-level.

# Translation

- High level language must be translated into a language the computer can understand





# How to translate?

**A program written in high-level programming language  
(for example, C program)**



**COMPILER (for example, gcc)**

**A low-level (machine language) program that is  
understandable by a computer (for example, a PC)**

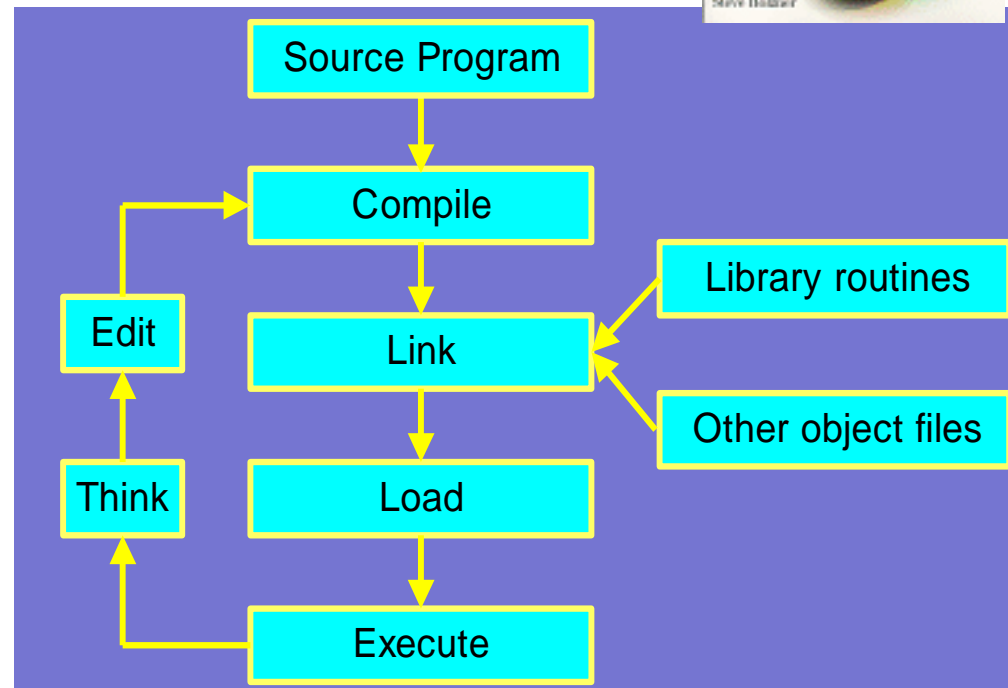
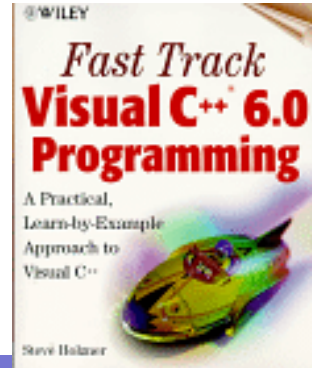
Examples of compilers:

- gcc, g++, Microsoft Visual C++

# Software Development

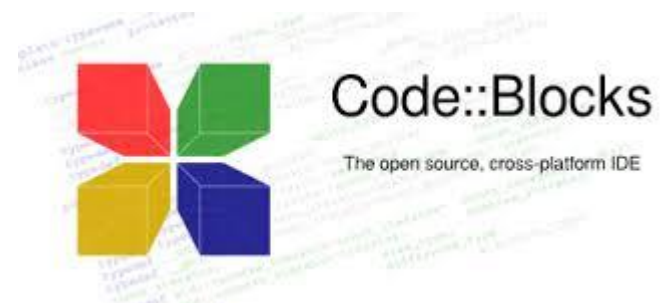
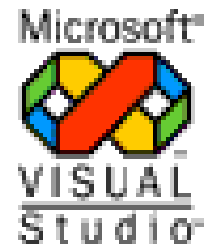
## ❖ Major activities

- Editing (writing the program)
- Compiling (creates .obj file)
- Linking with compiled files (creates .exe file)
  - Object files
  - Library modules
- Loading and executing
- Testing the program



# Integrated Development Environments

- Combine all of the capabilities that a programmer would want while developing software (VC++ 2008, Eclipse, CodeBlocks)
  - Editor
  - Compiler
  - Linker
  - Loader
  - Debugger
  - Viewer



# Writing Programs

- While writing a program we have to-
  - Understand requirements
  - Write an *algorithm*
  - *Implement* your algorithm
  - *Test* your code

# What is an algorithm?

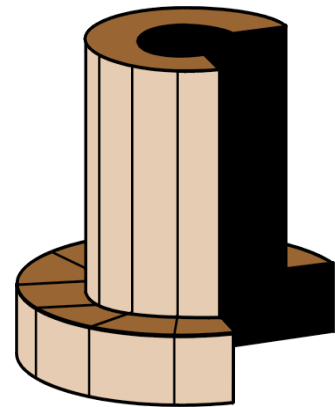
- A *specific* set of instructions
- An algorithm to make a cup of coffee?

# C Programming Language

- What is C?
  - C is a structured, relatively low-level, portable programming language.
- Why study C?
  - Many popular software tools are written in C.
  - First programming language that came up with many diverse features.
  - Has strongly influenced many other languages.
    - C-shell, java, C++, Perl, etc.
  - Forces the user to understand fundamental aspects of programming.
  - Very concise language.

# Most Important Feature of C

- Most important feature of C: its strong and efficient support of Structured Programming.
- C runs on: **All Computers**
  - PC
  - Macintosh
  - Unix workstations (also, Unix versions of C are free!)
  - supercomputers



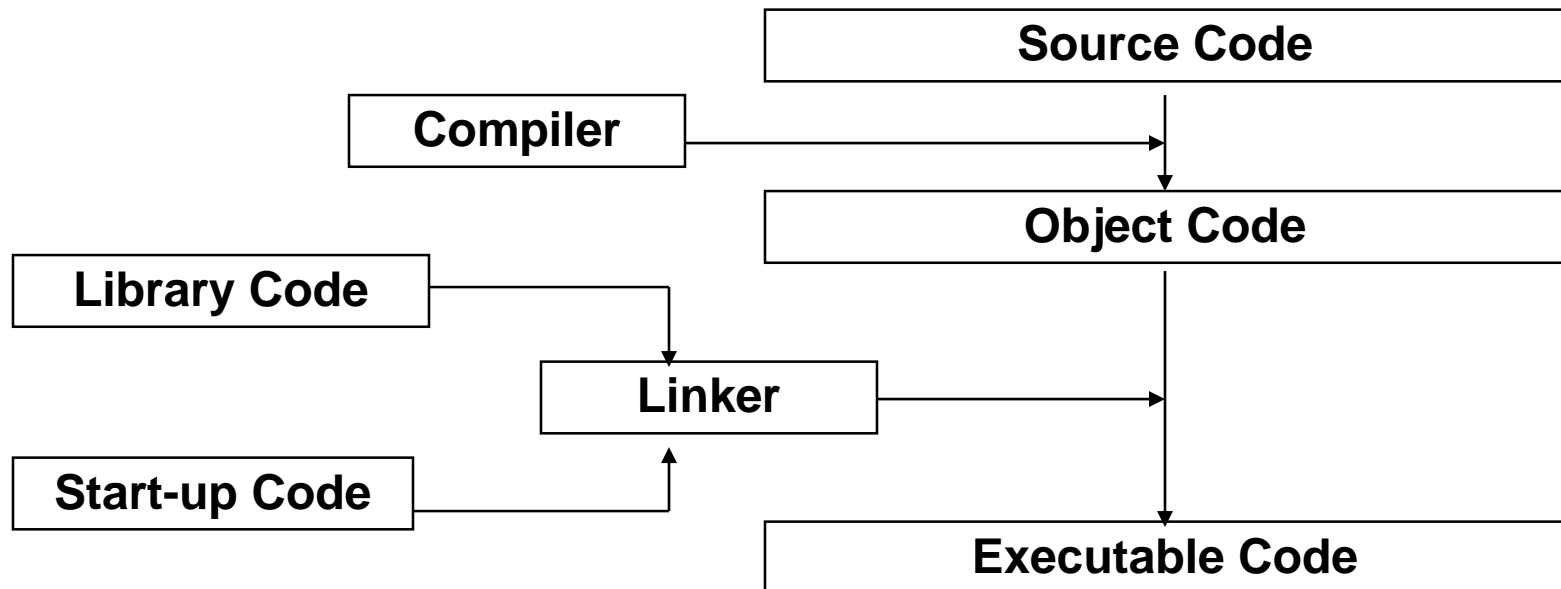
# Basics of a Typical C Environment

- C systems
  - Program-development environment
- C program names extensions
  - .c
- Executable file extensions
  - .exe



# C Basics

- **Case Sensitive**: UPPERCASE vs. lowercase
- **NOT Strongly typed**: Checking left to Programmer
- **Each Statement ends with a semicolon(;)**
- **Ellipses and Parentheses MUST match**: {.... } ( .... )
- Each program must have a function named **main**.
- Each functions scope is defined by {.... }
- Variables declared inside of {.... } is only visible to that scope.
- C is a translated language (vs. interpreted):



# C Basics Cont...

- A C program basically consists of the following parts –
  - Preprocessor Commands(begin with # symbol)
  - Functions(exactly one **main** function in each program)
  - Variables
  - Statements & Expressions
  - Comments(`/*.....*/` or `//.....` )

# C Basics Cont...

- Precompiler Instructions:

Additional code/shorthand/  
definitions

```
#include <stdio.h>
#define ZERO 0
(function prototypes)
(Global variables)
```

- Main Function:

A C program consists of **one or more function**. The primary function MUST be named **main**

```
(data type) main (arguments)
{
    declarations
    statements
    (return value)
} // end
```

- Additional Functions:

(optional) functions are  
the building blocks of C

**function 1**

**function n**

# C Basics Cont...

- Common Input/output functions
  - `scanf()`
    - Standard **input stream**
    - Normally keyboard
  - `printf()`
    - Standard **output stream**
    - Normally computer screen
  - `fprintf(stderr, "")`
    - Standard **error stream**
    - Display error messages

# C Basics Cont...

- Comments
  - Document programs
  - Improve program readability
  - Ignored by compiler
  - Single-line comment
    - Use C's comment `/* .. */` OR Begin with `//` **or**

# First C Program: Hello CSE 48!!

Left brace { begins function body.

```
/*  
 * Author Name:CSE-48  
 * Source file name: hello.c  
 * Comments: My first "hello, world" program in C  
 */  
#include <stdio.h>  
/*the main function*/  
int main ()  
{  
    /*print the phrase */  
    printf("Hello, CSE 48!!");  
    return 0;  
}
```

comments.

Preprocessor directive to include **standard input/output** stream header file **<stdio.h>**.

Function **main** returns an integer value.

Function **main** appears exactly once in every C program..

Statements end with a semicolon ;.

*printf* function prints the message on screen.

Keyword **return** is one of several means to exit function; value **0** indicates program terminated successfully.

Corresponding right brace } ends function body.

# C Basics Cont...

```
/*  
 * Author Name:CSE-48  
 * Source file name: hello.c  
 * Comments: My first program in C.  
 */
```

Comment

```
#include <stdio.h>
```

Preprocessor

```
/*the main function*/
```

Comment

```
int main ()
```

```
{
```

Main  
Function

```
/*print the phrase */
```

Comment

```
printf("Hello, CSE 48!!");
```

Function

```
return 0;
```

```
}
```

# Errors

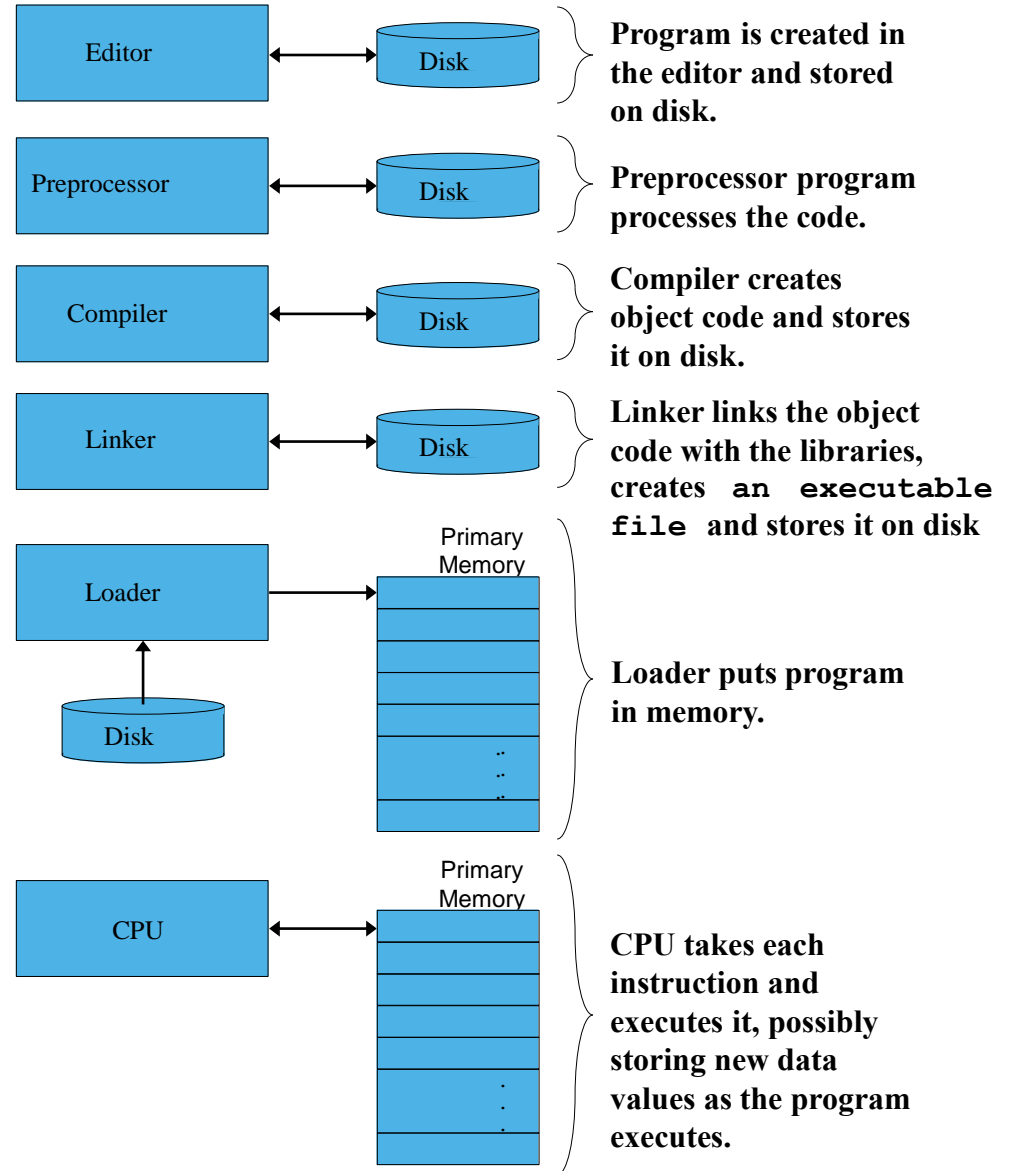
- You WILL have errors in your programs
- Syntax Errors
  - Compiler doesn't understand code
  - Ex. `–print("Hello, CSE 48!!!");`
  - Ex. `–printf("Hello, CSE 48!!!");`
  - Sometimes error messages don't match problem
- Logical Error
  - Program runs, but doesn't do what you want
  - Ex. `–printf("Hello, CSE!!!");`
  - Can be hard to track down



# Basics of a Typical C Environment

## Phases of C Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



# C: Dangers

- ▶ C is not object oriented!
  - Can't "hide" data as "private" or "protected" fields
  - You can follow standards to write C code that looks object-oriented, but you have to be disciplined – will the other people working on your code also be disciplined?
- ▶ C has portability issues
  - Low-level "tricks" may make your C code run well on one platform – but the tricks might not work elsewhere
- ▶ The compiler and runtime system will rarely stop your C program from doing stupid/bad things
  - Compile-time type checking is weak
  - No run-time checks for array bounds errors, etc. like in Java