Programming in C

Lecture Slides

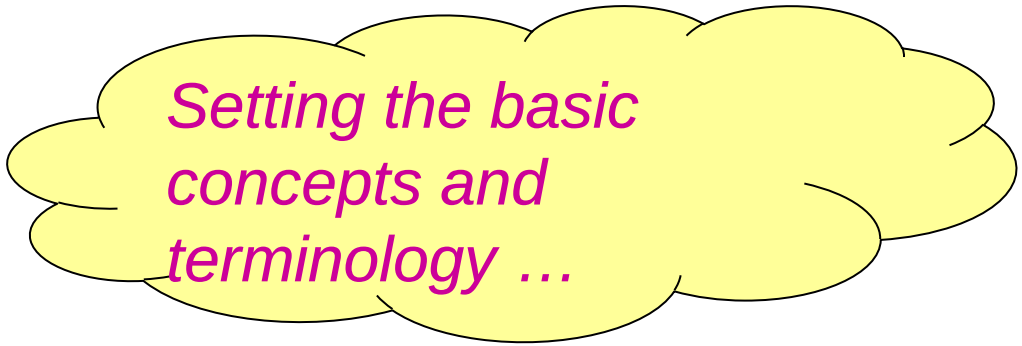
School of Electronics & Information Engineering
Taizhou University

Textbooks

- Stephen Prata, *C Primer Plus*, 6th Edition, Addison-Wesley, 2013
  - Main (first) textbook for this course
  - Teaches you how to program (in C)
- Brian Kernighan and Dennis Ritchie, *The C Programming Language*, 2nd Edition, Prentice Hall
  - Is considered “THE” book on C : coauthor belongs to the creators of the C programming language
  - The book is not an introductory programming manual; it assumes some familiarity with basic programming concepts

Fundamentals – Chapter outline:

- Classical model for computing machines
- Programming
- Programming languages
- Compiling
- Operating system

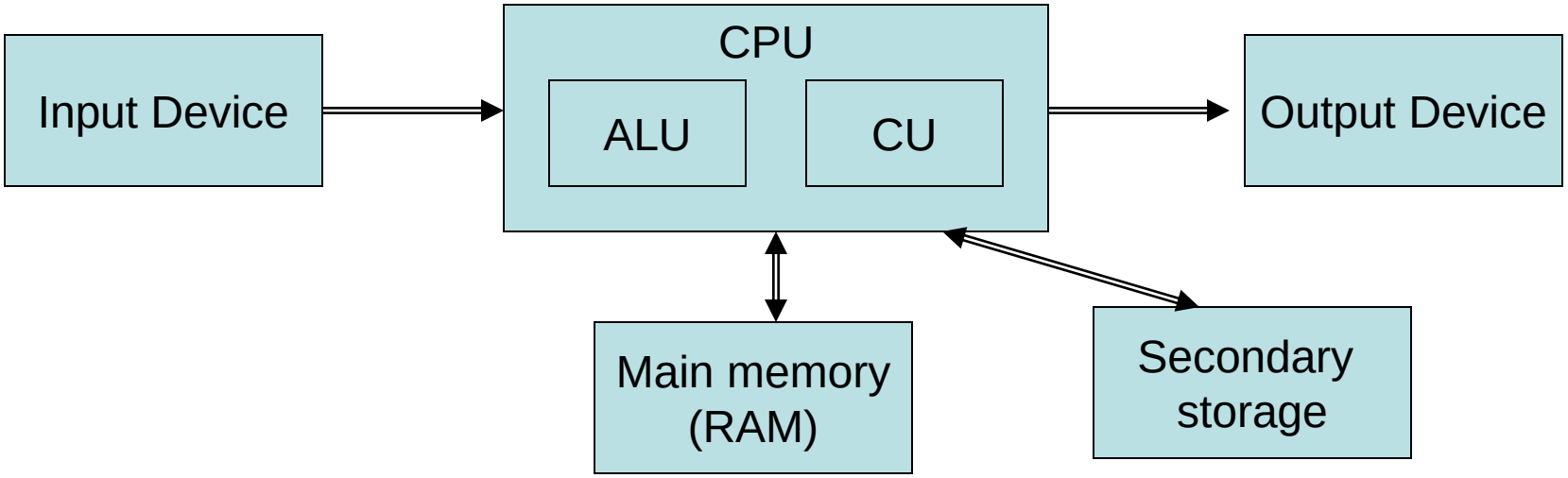


Model of a computing machine

- Computing machine (Computer): "*a machine that **stores** and **manipulates information** under the control of a **changeable program** that is **stored** in its memory*."

  - Pocket calculator: not a computer ! Manipulates information, but is built to do a specific task (no changeable stored program)
- This model is named the "von Neumann architecture" (John von Neumann – 1945; EDVAC - Electronic Discrete Variable Automatic Computer – the first stored-program computer)

The von Neumann architecture



The von Neumann architecture

- *Central Processing Unit* (CPU): the “brain” of the machine.
  - CU: Control Unit
  - ALU: Arithmetic and Logic Unit
    - Carries out all basic operations of the computer
    - Examples of basic operation: adding two numbers, testing to see if two numbers are equal.
- *Main memory* (called RAM for *Random Access Memory*): stores **programs and data**
  - Fast but volatile
- *Secondary memory*: provides permanent storage
- Human-computer interaction: through input and output devices.
  - keyboard, mouse, monitor
  - Information from input devices is processed by the CPU and may be sent to the main or secondary memory. When information needs to be displayed, the CPU sends it to the output device(s).

How it works

- How does a computer execute a program ? (example programs: a computer game, a word processor, etc)
- the instructions that comprise the program are copied from the permanent secondary memory into the main memory
- After the instructions are loaded, the CPU starts executing the program.
- For each instruction, the instruction is retrieved from memory, decoded to figure out what it represents, and the appropriate action carried out. (the *fetch- execute cycle)*
- Then the next instruction is fetched, decoded and executed.

Machine level programming

- Example: suppose we want the computer to add two numbers, and if the preliminary result is less than 10, then add 10 to the result

- The instructions that the CPU carries out might be :

```
[INSTR1]  Load into ALU the number from mem location 15
[INSTR2]  Load into ALU  the number from mem location 7
[INSTR3]  Add the two numbers in the ALU
[INSTR4]  If result is bigger than 10 jump to [INSTR6]
[INSTR5]  Add 10 to the number in the ALU
[INSTR6]  Store the result from ALU into mem location 3
```

- The **processors instruction set**: all basic operations that can be carried out by a certain type of processor

Machine level programming

- the **instructions and operands** are represented in *binary* notation (sequences of 0s and 1s).
  - Why binary ? Because computer hardware relies on electric/electronic circuits that have/can switch between 2 states
  - **bit** (binary digit)
  - Byte: 8 bits
- The program carried out by the CPU, on a hypothetical processor type, could be:

```
1010 1111
1011 0111
0111
…
```

- This way had to be programmed the first computers !
- The job of the first programmers was to code directly in machine language and to enter their programs using switches

Higher level languages

- Assembly language
  - First step from machine language
  - Uses ***symbolic names*** for operations

  - Example: a hypothetical assembly language program sequence:

```
1010 1111
1011 0111
0111
0011 1010
0010 1100
0110 1010
...
```

```
LD1  15
LD2   7
ADD
CMP  10
JGE  12
ADD 10
...
```

- Assembly language (cont)
  - Translation of assembly language into machine language: in the beginning done manually, later done by a special computer program – the *assembler*
  - Disadvantages: Low-level language:
    - programmer must learn the instruction set of the particular processor
    - Program must be rewritten in order to run on a different processor type – program is not *portable*

Higher level languages

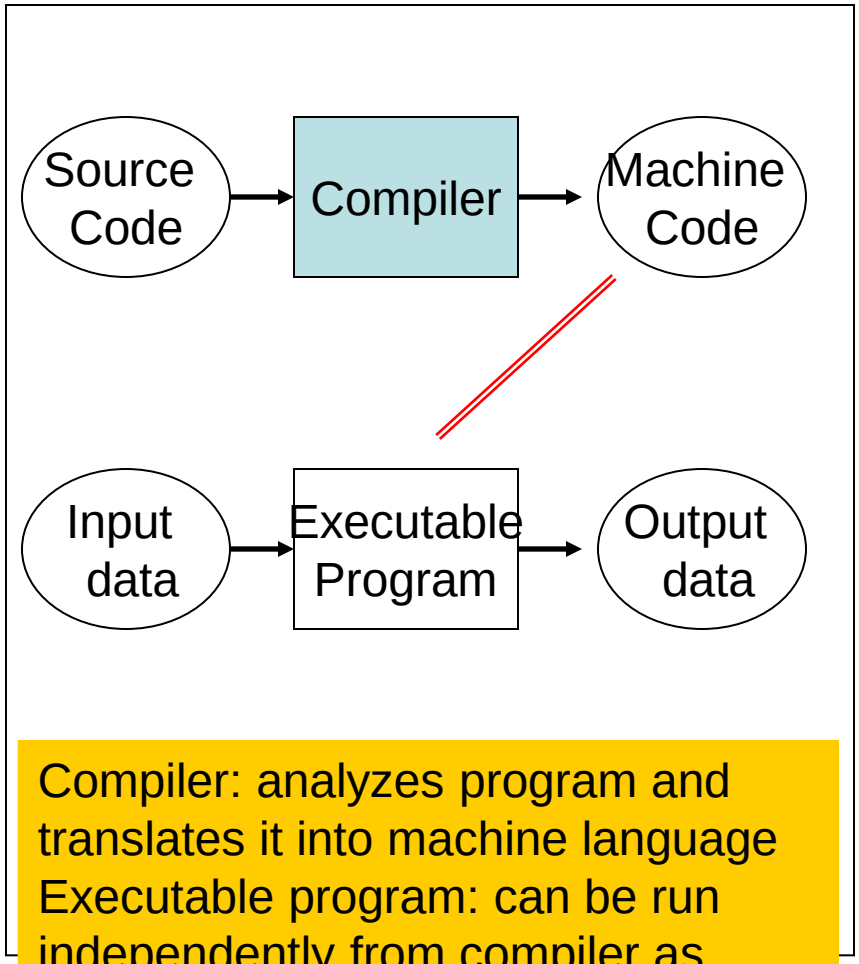
- High level languages
  - Using ***more abstract instructions***
  - **Portable** programs result

  - Example: a hypothetical program sequence:

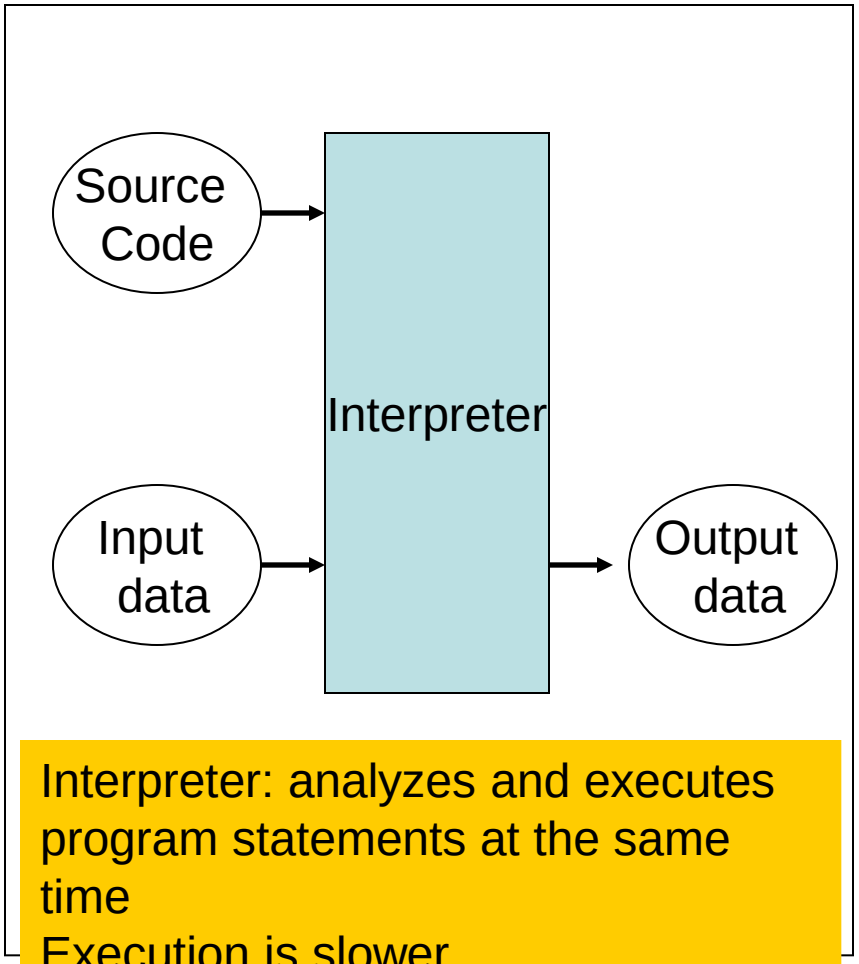
```
DEFVAR a,b,c;
BEGIN
        READ a
        READ b
        READ c
        c := a+b
        IF (c <10) THEN c:=c+10
        PRINT c
END                              …
```

- High level languages
  - Writing portable programs, using more abstract instructions
  - A high level instruction (*statement*) is translated into many machine instructions
  - Translation of high level language into machine instructions: done by special computer programs – *compilers* or *interpreters*

Compilers/Interpreters



Compiler: analyzes program and translates it into machine language
Executable program: can be run independently from compiler as many times => fast execution



Interpreter: analyzes and executes program statements at the same time
Execution is slower
Easier to debug program

Operating Systems

- Operating system: a program that controls the entire operation of a computer system:
  - Handles all input and output (I/O) operations that are performed on a computer
  - manages the computer system’s resources
  - handles the execution of programs (including multitasking or multiuser facilities)
- Most famous OS families:
  - Windows
  - Unix

Higher Level Languages

- Programming Paradigms:
  - Imperative Programming: describes the exact sequences of commands to be executed
    - Structured programming, procedural programming
      - FORTRAN, C, PASCAL, …
    - Object oriented programming
      - **C++, Java, C#, …**
  - Declarative programming: program describes *what* it should do, *not how*
    - Functional programming
      - Lisp, ML, …
    - Logic Programming
      - Prolog

Why C

- C is an efficient language.
- C is a portable language
- C is powerful and flexible language
- C is Programmer Oriented language
- why we're learning C not C++

The C Programming Language

- Developed by Dennis Ritchie at AT&T Bell Laboratories in the early 1970s
- Growth of C tightly coupled with growth of Unix: Unix was written mostly in C
- Success of PCs: need of porting C on MS-DOS
- Many providers of C compilers for many different platforms => need for standardization of the C language
- 1990: ANSI C (American National Standards Institute)
- International Standard Organization: ISO/IEC 9899:1990
- 1999: standard updated: C99, or ISO/IEC 9899:1999

The first C program

uses standard library input and output functions (printf)

the program

begin of program

statements

end of program

```
#include<stdio.h>
int add(int x,int y) // Calculate the sum of x, y
{
    int z;
    z = x + y;
    return z; /* Returns the value of the
accumulated sum to the caller */
}
```

main begin execution. It is a spe

first s e string of characters "Prog

last s system a status value of 0 (

```
void main()
{ int  i1, i2, sum;
  printf("Enter two integer numbers:\n");
  scanf("%d,%d", &i1, &i2);
  sum = add(i1, i2); // Call function calculation add
  printf("sum=%d\n", sum);
```

Using comments in a program

- Comment statements are used in a program to document it and to enhance its readability.
- Useful for human readers of the program – compiler ignores comments
- Ways to insert comments in C:
  - When comments span several lines: start marked with /*, end marked with */
  - Comments at the end of a line: start marked with //

Using comments in a program

```
/* This program adds two integer values
and displays the results */

#include <stdio.h>
int main (void)
{
    // Declare variables
    int value1, value2, sum;
    // Assign values and calculate their sum
    value1 = 50;
    value2 = 25;
    sum = value1 + value2;
    // Display the result
    printf ("The sum of %i and %i is %i\n",
                                        value1, value2, sum);
    return 0;
}
```

The first C program

uses standard library input and output functions (printf)

the program

begin of program

statements

end of program

```
#include<stdio.h>
int add(int x,int y) // Calculate the sum of x, y
{
    int z;
    z = x + y;
    return z; /* Returns the value of the
accumulated sum to the caller */
}
```

main begin execution. It is a spe

first s e string of characters "Prog

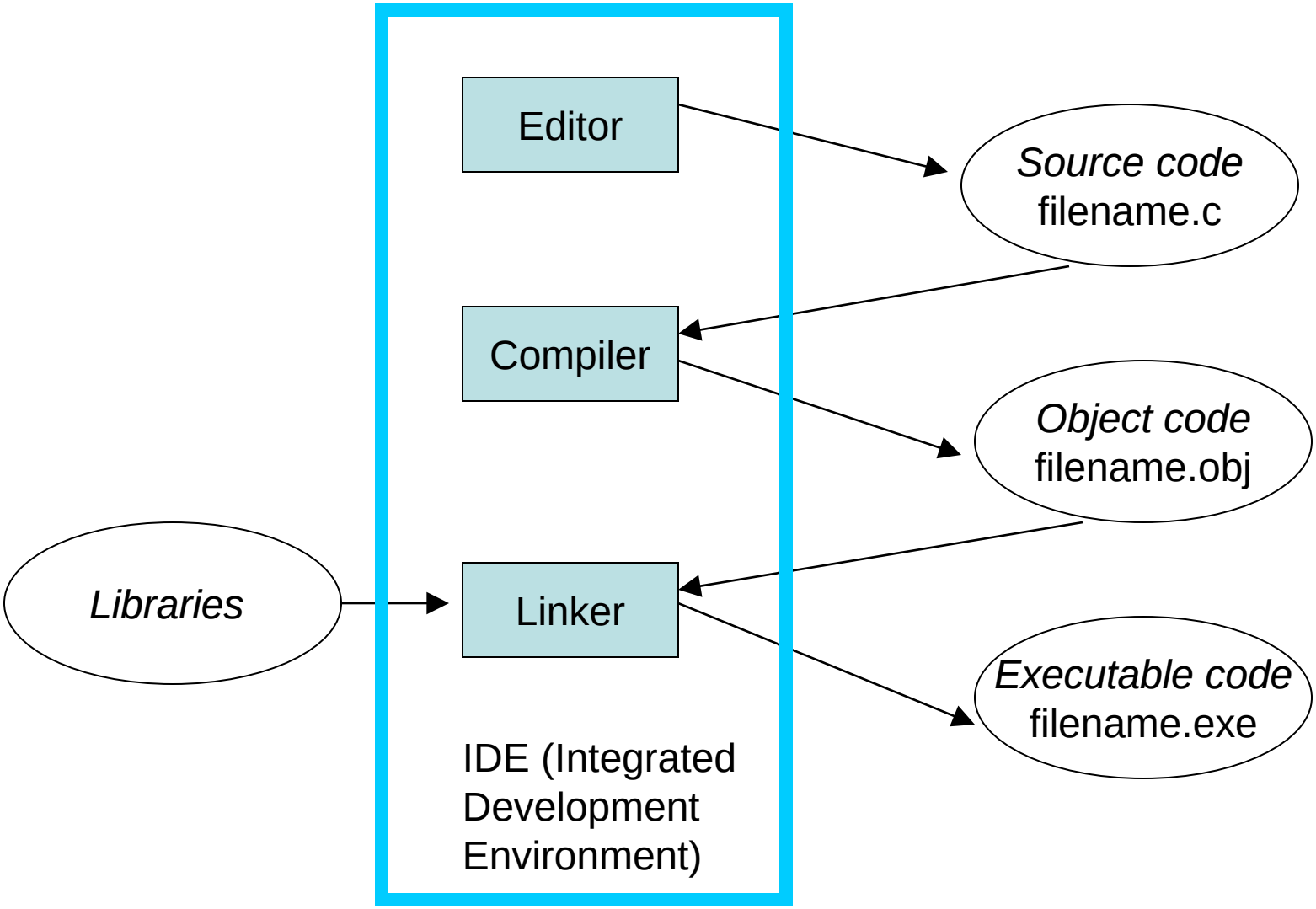
last s system a status value of 0 (

```
void main()
{ int  i1, i2, sum;
  printf("Enter two integer numbers:\n");
  scanf("%d,%d", &i1, &i2);
  sum = add(i1, i2); // Call function calculation add
  printf("sum=%d\n", sum);
```

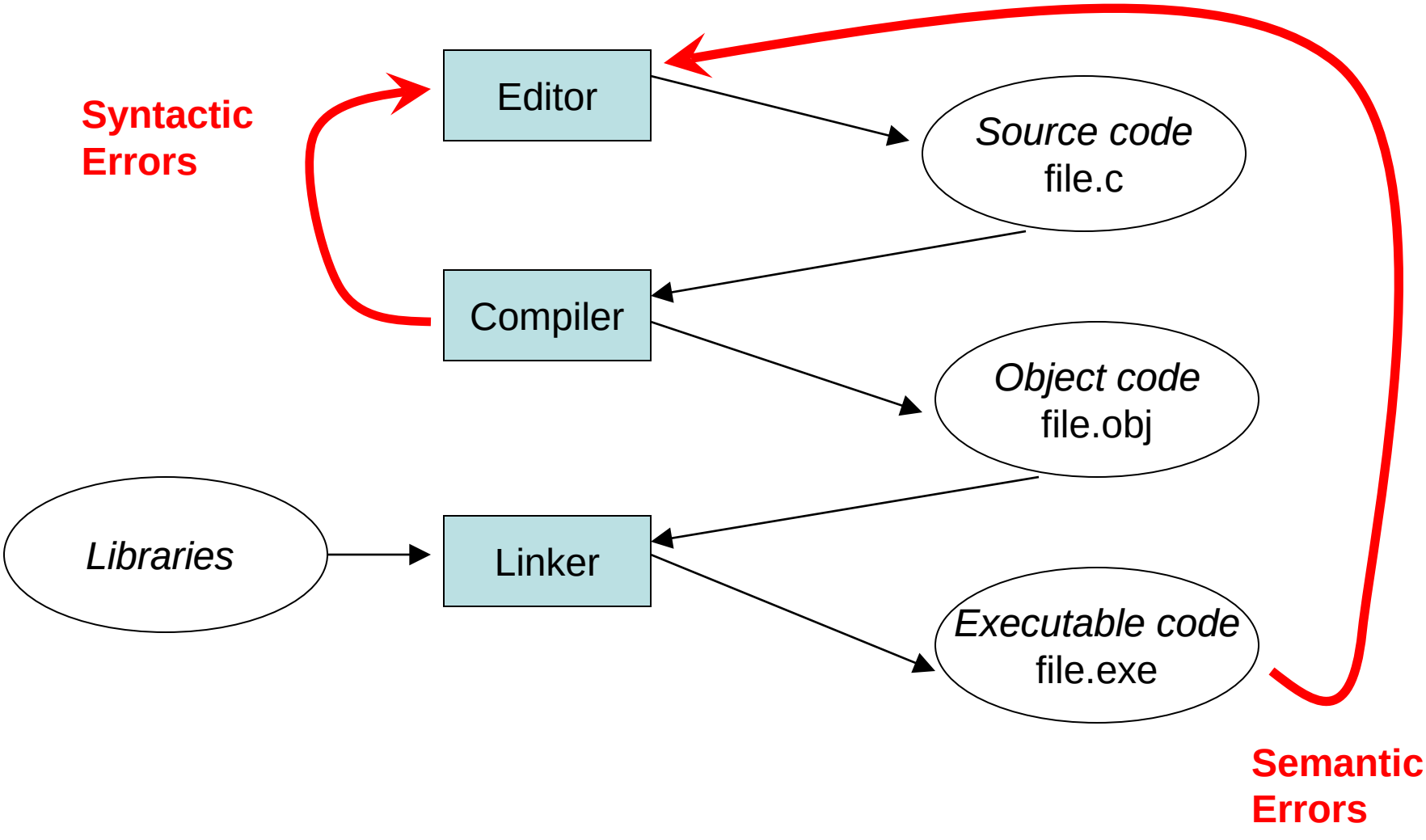
The format in C

- Statements are terminated with semicolons
- Indentation is nice to be used for increased readability.
- Free format: white spaces and indentation is ignored by compiler
- **C is case sensitive** – pay attention to lower and upper case letters when typing !
  - All C keywords and standard functions are lower case
  - Typing INT, Int, etc instead of int is a compiler error
- Strings are placed in double quotes
- New line is represented by \n (Escape sequence)

Compiling and running C programs



Debugging program errors



Displaying multiple lines of text

```
#include <stdio.h>
int main (void)
{
   printf ("Testing...\n..1\n...2\n....3\n");
   return 0;
}
```

It is not necessary to make a separate call to printf for each line of output !

Output:

```
Testing...
..1
...2
....3
```