C Programming Language

(1st class)

Dohyung Kim

Assistant Professor @ Department of Computer Science

Welcome

- **This is C Programming Language course**

- **What is this course about**
  - C language syntax
  - Basic Programming Skills

Today …

- Introduction to this course
- Introduction to Computer and Programming
- Why do we learn C?
- Preliminaries

Administrative Information

- **Class Room**
  - #412 (lecture) and #309 (practice) in Hanbit Bldg.
- **Class Hour**
  - Lecture : Tue 2nd, 3rd
  - Practice : Tue 5th , 6th
- **Office hours**
  - With prior reservation via email : mr.dhkim@gmail.com

Lecturer

- **Dohyung Kim**
  - Assistant Professor, Department of Computer Science
  - Email - mr.dhkim (at) gmail.com
  - Office - #401, Hanbit Bldg.
  - Website - https://sites.google.com/view/icnlab

- **TAs**
  - Will be updated

Textbook & References

- **No official textbook**
  - There are lots of stuffs on the Internet

Course Components

- **Lectures**
  - Concepts
  - Backgrounds
- **Lab Sessions**
  - Practice what you've learned

Grading Policy (Tentative)

- Mid-term : 30%
- Final-term : 30%
- Programming Assignments : 30%
- Attendance & Quiz : 10%

You will fail this course when

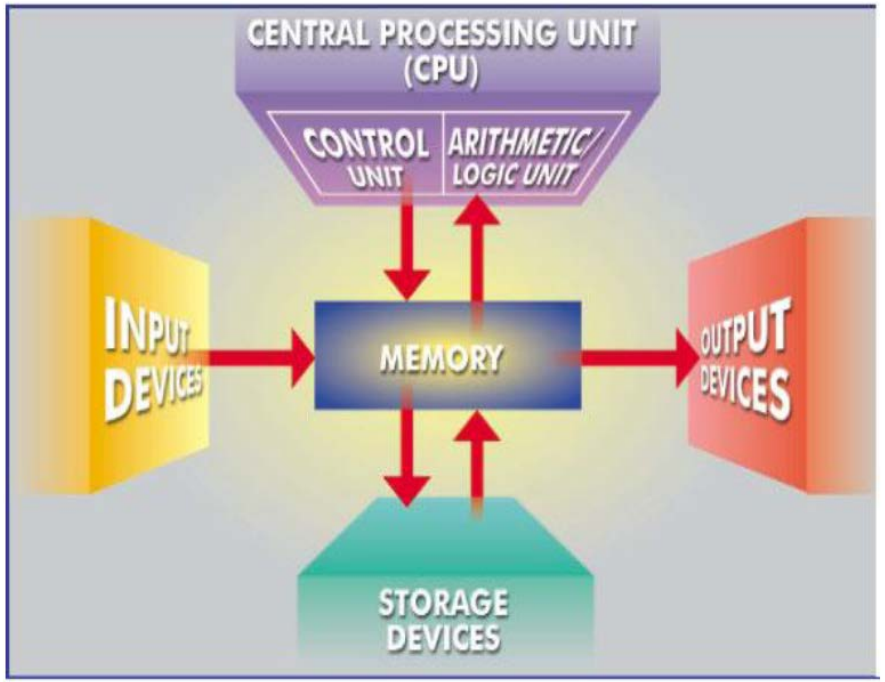
1) Missing more than 4 class hours
2) Cheating in assignments or exams

What is computer

- **A computer is an electronic device that can**
  - **Receive** information
  - **Perform** processes
  - Produce **output**
  - **Store** info for future use

Primary components of a computer

- **Central Processing Unit**
  - control unit and arithmetic/logic unit
- **Memory (Main Memory)**
- **Input devices**
- **Output devices**
- **Storage devices**



Memory (Main Memory)

- Working area
- Temporarily stores program and data (while program is executing)
- Data is stored in the form of bit/byte in the memory.

* Bit = one binary digit (0 or 1)
* Byte = 8 bits
* Memory is a collection of spaces that contain one byte-sized data
* Pieces of data may have different sizes

char arr[3] = {‘a’, ‘b’, ‘c’}

Address	Data Byte	
3021	1111 0000	Item 1: 2 bytes stored
3022	1100 1100	
3023	1010 1010	Item 2: 1 byte stored
3024	1100 1110	Item 3: 3 bytes stored
3025	0011 0001	
3026	1110 0001	
3027	0110 0011	Item 4: 2 bytes stored
3028	1010 0010	
3029	…	Next Item, etc.

Software

- **Collection of instructions that enables a user to interact with the computer**
  - System software
    - Operating Systems (Windows, DOS, Apple, Unix, Linux)
  - Application software
    - Word processing, Electronic spreadsheet, Database, Presentation graphics

Programming

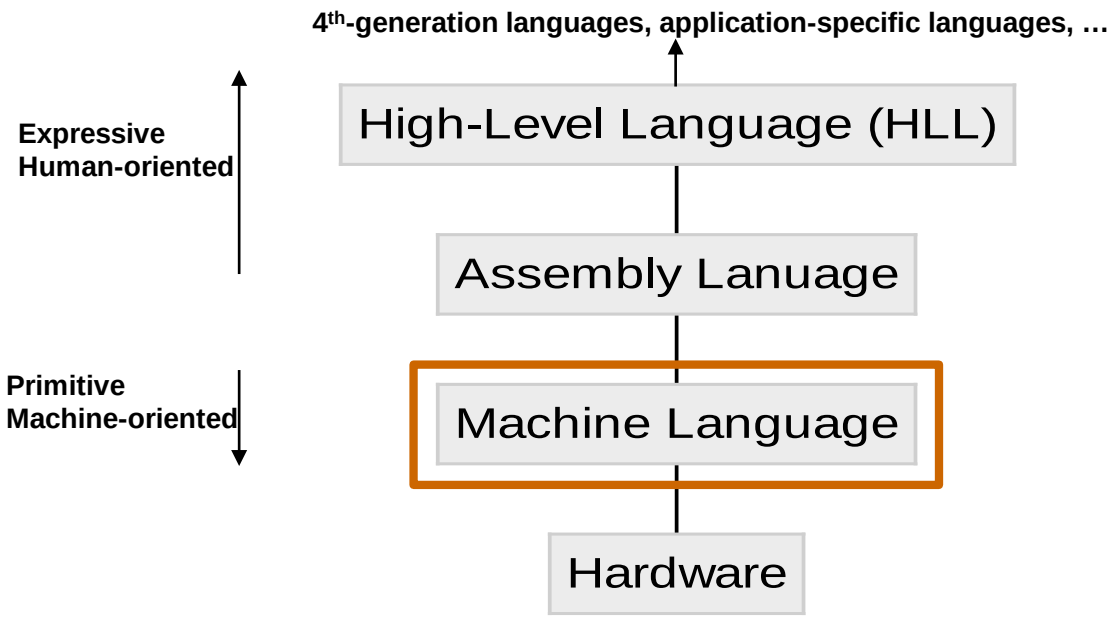
- Write a series of instructions in a way the computer can understand

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

- Computers may understand, but …

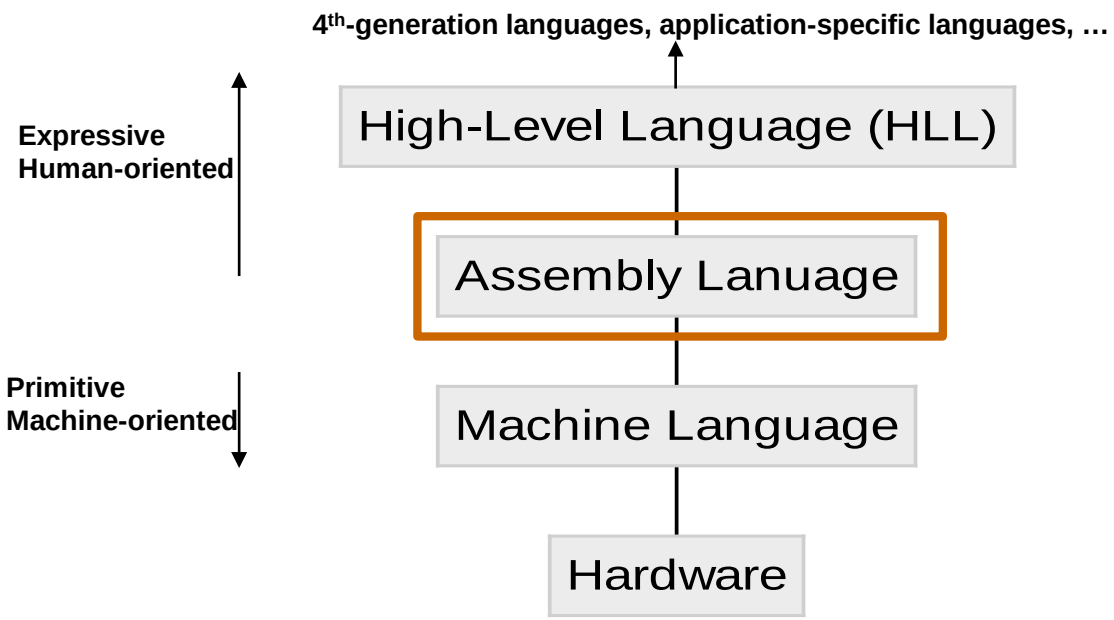
- More user-friendly languages have been designed for programming

Programming Language




```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

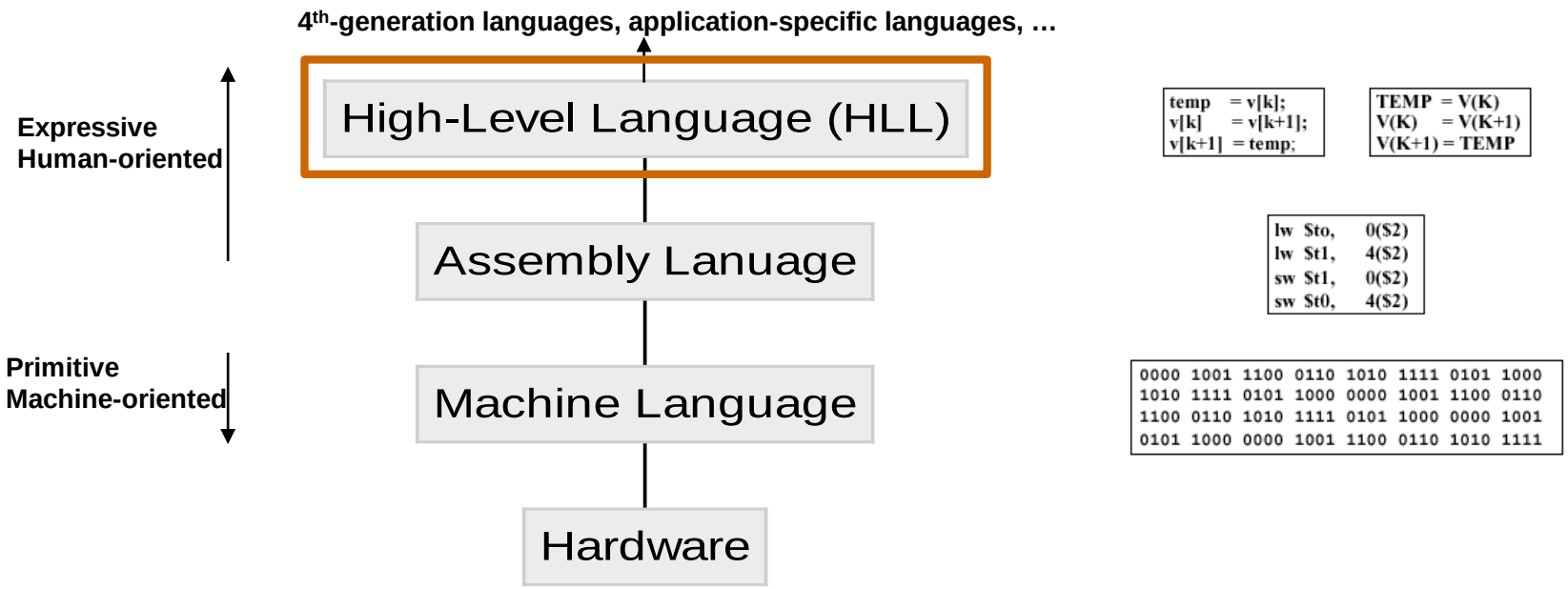
Programming Language




```
lw  $to,    0($2)
lw  $t1,    4($2)
sw  $t1,    0($2)
sw  $t0,    4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Programming Language




```
temp   = v[k];
v[k]   = v[k+1];
v[k+1] = temp;
```

```
TEMP   = V(K)
V(K)   = V(K+1)
V(K+1) = TEMP
```

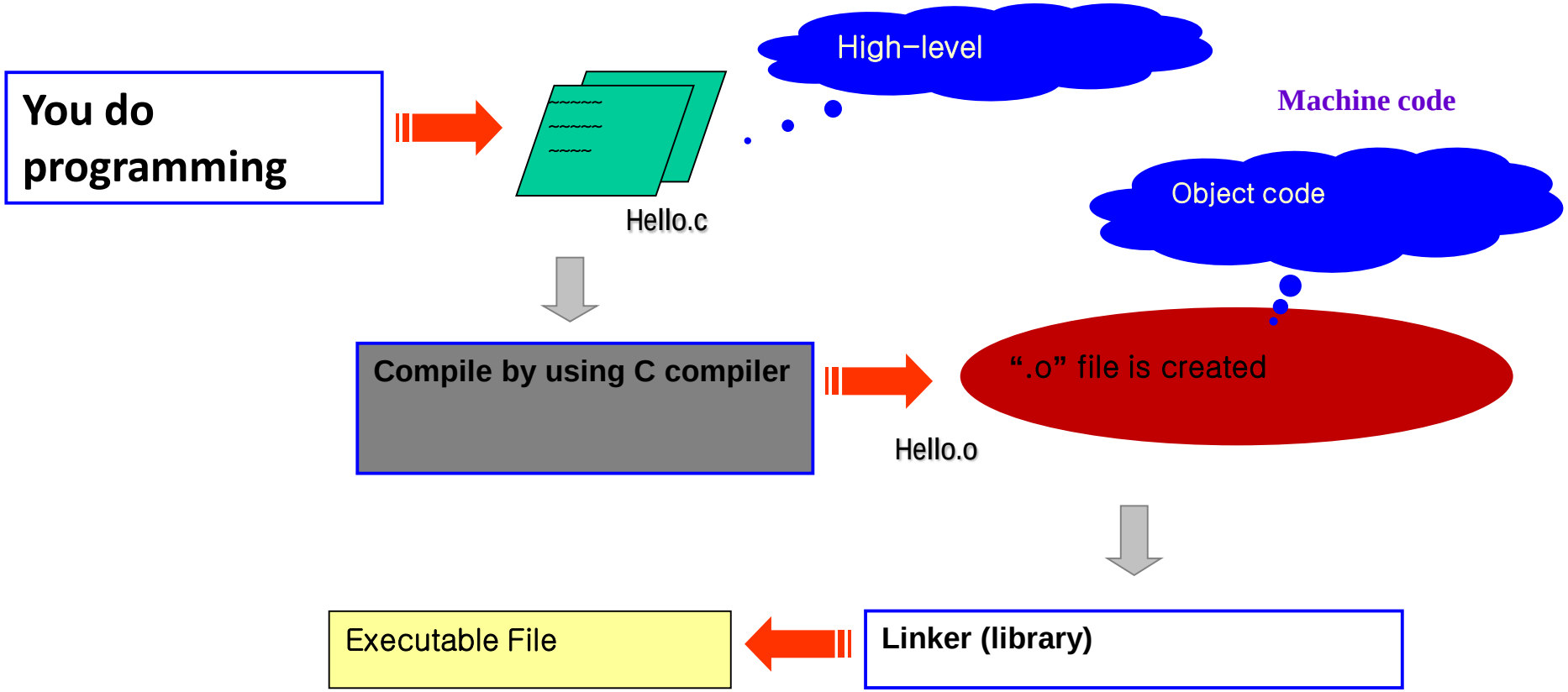
```
lw  $to,   0($2)
lw  $t1,   4($2)
sw  $t1,   0($2)
sw  $t0,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Compile/Interpreter

- User-friendly languages should be translated into the machine code (Otherwise, how would the machine do work?)
- Compile/Interpreter

Compile


You do programming
Hello.c
High-level
Machine code
Object code
Compile by using C compiler
Hello.o
“.o” file is created
Executable File
Linker (library)

Compilers vs. Interpreters

- **Compilers**
  - Fix decision that can be taken at compile time
    - Error checking, Static allocation, Code optimization
  - Compilation leads to better performance in general

- **Interpreters**
  - Facilitates interactive debugging and testing
    - Procedures can be invoked from command line by a user
    - Variable values can be inspected and modified by a user

Comparison btw. Compiler and Interpreter

	Compiler	Interpreter
Input	It takes an entire program at a time.	It takes a single line of code or instruction at a time.
Output	It generates intermediate object code.	It does not produce any intermediate object code.
Working mechanism	The compilation is done before execution.	Compilation and execution take place simultaneously.
Speed	Comparatively faster	Slower
Memory	Memory requirement is more due to the creation of object code.	It requires less memory as it does not create intermediate object code.
Errors	Display all errors after compilation, all at the same time.	Displays error of each line one by one.
Error detection	Difficult	Easier comparatively
Pertaining Programming languages	C, C++, C#, Scala, …	PHP, Perl, Python, Ruby …

Write a Program

- Decide what step are needed to complete the task
- Write the steps in pseudocode (Written in English generally) or as a flow chart
- Translate into the programing language
- Try out the program and “debug” it if necessary

Pseudocode

- **Like the instructions for a recipe**
- **List of step written in English**
- **Must be in the right sequence**
  - Bake the cake after mixing it up?
- **Example of the program that adds two numbers**
  - Start
  - Get two numbers
  - Add them
  - Print the answer
  - End

Why Do We Learn C?

- **Why C, and not assembly language?**
  - Universality and portability across various computer architectures
    - Palm OS cobalt smartphone (ARM processor), original iMac (PowerPC), Arduino (Atmel AVR), Intel iMac(Intel Core 2 Duo)
    - Assembly languages are machine-dependent, completely incompatible with each other.
  - Hard to read or interpret in a logical way
- **Why C, and not another language?**
  - Performance

Say Hello

```
#include<stdio.h>
int main ( ){
   printf(“Hello, everyone”);
   return 0;
}
```

1: #include<stdio.h>
- header file which includes the printf( ) function as well as others

2: int main ( ){
- the standard expected function name “main”. Every C program starts from the main function
- the type of return value : int (integer) , no input parameters

3: printf(“Hello, everyone”);
- “Hello, everyone” is displayed on the monitor

4: return 0;
- Integer value 0 is returned after the function ends

Preliminaries

- **Statement**
  - is text which is turned into executable instruction

```
int i = 6;
printf("hello");
```

- **Block Structure**
  - Consists of executable statements
  - Begins with an opening brace "{" and ends with a closing brace "}"
  - Readability and scope

```
int main (void)
{
        int i = 5;
        {
                int i = 6;
        }
        return 0;
}
```

Preliminaries (cont'd)

- **White space**
  - Refers to the tab, space, and newline/EOL (End Of Line) characters that separate the text characters that make up source code lines.
  - is used for human readability in source code.
  - The compiler simply skips over whitespace.

```
int i = 6;  printf("hello");
```

```
int i = 6;
printf("hello");
```

```
int i = 6;
Printf (
"hello");
```

Preliminaries (cont'd)

- **Scope**

```
int i = 5;
int main (void)
{
        int i = 5;
        printf ("%d\n", i);
        return 0;
}
```

```
int main (void)
{
        int i = 5;
        {
                int i = 6;
                printf ("%d\n", i);
        }
        printf("%d\n", i);
        return 0;
}
```

Preliminaries (cont'd)

- **Function**
  - A special kind of block that performs a well-defined task
  - Function call (e.g. printf(…);)
    - You don't need to know how the function is written
    - But you need to know
      - What the function do
      - The data type of the arguments and what they mean
      - The data type of the return value and what it means
  - Keep in mind that every executable program needs to have **one, and only one, main function** which is where the program begins executing

Preliminaries (cont'd)

- **The Standard Library**
  - A basic set of functions common to each implementation of C
  - Provides functions for tasks such as input/output, string manipulation, mathematics, files and memory allocation.
  - #include<stdio.h>, #include<stdlib.h>,…

Remarks

- Concepts of High-level/Low-level languages
- Compile/Interpreter
- C Preliminaries

Q and A