Lab #1. Bit Lab

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

About the Labs

- **Labs will count for 25% of the total score in semester**
- **We will have three lab assignments**
  - **Lab #1:** Bit Lab (5%)
  - **Lab #2:** Reversing Lab (10%)
  - **Lab #3:** Cache Lab (10%)
- **Today: Lab #1 (Bit Lab)**
  - Three small C programming exercises
  - Puzzles using *bit-level* operations (a.k.a. *Data Lab* in *CSAPP*)
- **Another goal of this lab is getting familiar with Linux and learning how to work with the skeleton code**

Remind: LLM (ChatGPT) Policy

- **Remember that it is NOT allowed to use ChatGPT or other LLMs in the lab assignment**
- **Still, you may feel tempted to do that**
  - Indeed, Lab #1 problems are relatively easy and you can quickly get the solution from ChatGPT (or by *Googling)*
- **But remember: if you start relying on ChatGPT from now on, it will eventually limit your capability**
  - You lose chance to practice and improve your skills, insights, etc.
- **If you keep working on challenging problems on your own, you will surpass ChatGPT quite soon**

General Information

- **Check the *Assignment* tab of *Cyber Campus***
  - Skeleton code (**`Lab1.tgz`**) is attached together with this slide
  - Submission will be accepted in the same post, too
- **Deadline: 4/3 Thursday 23:59**
  - Late submission deadline: **4/5 Saturday 23:59 (-20% penalty)**
  - Delay penalty is applied uniformly **(not problem by problem)**
- **Please read the instructions in this slide carefully**
  - This slide is a step-by-step tutorial for the lab
  - It also contains important submission guidelines
    - If you do not follow the guidelines, **you will get penalty**

Skeleton Code Structure

- **Copy `Lab1.tgz` into CSPRO server and decompress it**
  - Recommend to use cspro2.sogang.ac.kr (**Ubuntu 20.04**)
  - Don't decompress-and-copy; copy-and-decompress
- **`1-1~1-3`: Each directory contains a problem**
- **`validate`: Verifies if your code satisfies the constraints**
- **`check.py`: Script for self-grading (explained later)**
- **`config`: Used by grading script (you may ignore)**

```
jason@ubuntu:~$ tar -xzf Lab1.tgz
jason@ubuntu:~$ ls Lab1/
1-1  1-2  1-3  check.py  config  validate
```

Problem Directory (Example: 1-1)

- **`bitMask.c`: This is the only file that you have to fill in**
  - **Do not make any modification other files**
- **`main.c`: This program will test your `bitMask.c` code**
- **`Makefile`: You can build the program by typing `make`**
  - If you have not heard of **`make`** or **`Makefile`**, take a brief look at makefiletutorial.com/
- **`testcase`: Contains test cases and expected outputs**

```
jason@ubuntu:~/Lab1/1-1$ ls
bitMask.c  main.c  Makefile  testcase
jason@ubuntu:~/Lab1/1-1$ ls testcase/
ans-1  ans-2  tc-1  tc-2
```

Tasks to do

- **For each problem, you have to implement a function**
  - **Read the comment in each file carefully:** it tells you what to do, provides examples and specifies assumptions on inputs
- **Problem 1-1 (`bitMask.c`)**
  - **`bitMask(x)`**: return a mask that has **`32-x`** 0's followed by **`x`** 1's
- **Problem 1-2 (`absVal.c`)**
  - **`absVal(x)`**: return the absolute value of **`x`**
- **Problem 1-3 (`conditional.c`)**
  - **`conditional(x, y, z)`**: return **`z`** if **`x`** is 0, return **`y`** otherwise

Constraints

- **There are some constraints that your code must satisfy**
  - If your code does not satisfy them, you will get 0 point
  - Allowed operators: `! ~ & ^ | + << >>`
    - Do **NOT** use other operators such as `&& || - == < > ?`
  - Use `int` type only
    - Do **NOT** use other primitive types or structure, array, etc.
  - Write **straight-line code**
    - Do **NOT** use any control constructs such as `if`, `do`, `while`, `for`, `switch`, etc.
  - Do **NOT** define or call any additional function
  - Also, do **NOT** include any header like `#include <stdio.h>`

Using the Validator

- **You can use `validate` to confirm whether your code satisfies the previous constraints**
  - It will print illegal points in the code you wrote
  - If it does not print anything, your code passed the validation

```
jason@ubuntu:~/Lab1$ cat 1-2/absVal.c
int absVal(int x) {
  if (x > 0)
    return x;
  else
    return -x;
}
jason@ubuntu:~/Lab1$ ./validate 1-2/absVal.c
dlc:1-2/absVal.c:2:absVal: Illegal operator (>)
dlc:1-2/absVal.c:5:absVal: Illegal operator (-)
dlc:1-2/absVal.c:5:absVal: Illegal if
```

Running Test Cases

- **Once you compile the program by typing `make`, you can run it by providing the path of test case file**
- **Some test cases and their expected outputs are already provided in the `testcase/` directory**
  - Output of running **`tc-N`** must match with **`ans-N`**

```
jason@ubuntu:~/Lab1/1-1$ make
gcc bitMask.c main.c -o main.bin
jason@ubuntu:~/Lab1/1-1$ cat testcase/tc-2
31
jason@ubuntu:~/Lab1/1-1$ cat testcase/ans-2
0x7fffffff
jason@ubuntu:~/Lab1/1-1$ ./main.bin testcase/tc-2
0x7fffffff
```

Self-Grading

- **Once you think everything is done, run `check.py` to confirm that you pass all the provided test cases**
  - Each character in the result has following meaning:

    '**O**': correct, '**X**': wrong,

    '**C**': compile error, '**T**': timeout

    '**I**': failed to pass the validator, '**E**': runtime error
  - So you must ensure that **`./check.py`** prints '**O**' for all the cases

```
jason@ubuntu:~/Lab1$ ./check.py
[*] 1-1: OO
[*] 1-2: II
[*] 1-3: XX
```

Test Cases for Grading

- **I will use different test case set to grade your code**
  - This means even if you pass all the provided test cases, it does not guarantee that you will get 100 pt.
- **So you are encouraged to run your code with your own test cases (try to think of various inputs)**
- **Some students ask me to provide more test cases, but it is important to practice this on your own**

Problem Information

- **Three problems in total**
  - Problem 1-1: **30 pt.**
  - Problem 1-2: **35 pt.**
  - Problem 1-3: **35 pt.**
- **You will get the point for each problem based on the number of test cases that your code passes**
  - Ex) If you pass half of the test cases during the grading, you will get 50% of the point

Submission Guideline

- **You should submit three C source files**
  - Problem 1-1: **`bitMask.c`**
  - Problem 1-2: **`absVal.c`**
  - Problem 1-3: **`conditional.c`**
- **If the submitted file does not compile by typing "`make`" command, cannot give you any point for that problem**
- **Submission format**
  - Upload these files directly to *Cyber Campus* (**do not zip them**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**