# 시스템 프로그래밍

## 3주차: 실습 프로젝트 - Datalab

# T A

강범우, IT/BT 701호

qjadn0630@hanyang.ac.kr

# Datalab 과제

- 과제의 목표
  - 코딩을 통해 정수와 부동소수점의 비트 수준 표현과 연산 이해

- 과제 수행 환경
  - C 언어와 gcc 컴파일 환경 (make 명령)
  - 필수 자료: datalab-handout.tar
    - 각자 본인의 gitlab 프로젝트 "Week3" 폴더에 업로드 될 예정

- 결과 제출 방법 및 기한
  - **bits.c** 소스 코드만 수정하여 gitlab 프로젝트에 commit
  - 9월 22일(화) 23:59까지 commit된 것만 인정

# Make 설치 참고

- Window
  - https://ndb796.tistory.com/381
  - VMWare, VirtualBox, …
- Linux
  - $ sudo apt-get install build-essential
- Mac OS
  - Install Homebrew
  - $ xcode-select --install

# 과제 수행 방법

- 필수 자료 확인
  - **datalab-handout.tar**
    - $ tar xvf datalab-handout.tar
    - 임의 프로그램 사용 가능
    - 과제 결과를 직접 테스트할 수 있는 소스와 프로그램 포함
- **“bits.c”** 파일 만 수정하여 과제 수행
  - 다른 파일은 절대 수정하지 말 것.

# 과제 수행 방법

- 과제 파일 (bits.c)
  - 세부 과제 함수의 목적, 코딩 제약, 예시, 배점 등 확인
  - 함수의 내용(body) 부분을 코딩함.
- 프로그래밍 기본 제약 사항
  - 32bit 시스템에서 실행되는 것을 전제로 한다.
    - Int의 크기는 4byte이며, 컴파일 시 –m32 옵션이 사용된다.
  - 허용된 연산자를 제한된 개수 이내만 사용한다.
    - bits.c 소스 내의 각 함수 설명 참고

# 주요 코딩 규칙 요약

- 정수형 과제 코딩 규칙
  - **정수형 상수는 0 ~ 255 (0xFF) 만 사용**
  - int (32bits) 데이터형만 사용
  - 사용 가능 연산자: ! ~ & ^ | + << >>
  - 조건문, 순환문 사용 금지
  - 매크로와 추가의 함수 사용 금지
  - 캐스팅(casting) 연산 사용 금지
- 부동소수점 과제 코딩 규칙
  - float 데이터형 사용 금지
  - 배열, struct 등의 복합 자료구조 사용 금지
  - 32비트 상수, 조건문, 순환문 사용 가능

# 과제 결과 확인 방법

- 과제 파일 컴파일
  - make 명령 실행
    - Makefile 에 정의된 규칙에 따라 자동으로 컴파일 됨.
    - bits.c 를 수정하는 경우 반드시 새로 make 해야 함.
- 점수 확인
  - btest 프로그램 실행
    - 각 과제 함수의 정상 동작 여부 점검 및 정확성 점수부여
  - dlc 프로그램 실행 (datalab compiler)
    - 프로그래밍 제약 위반 사항 확인 (허용되지 않은 연산자 등)
  - driver.pl 스크립트 실행
    - 과제의 총 점수 평가 (정확성, 성능 점수 합계)

# 과제 내용 (bits.c)

- 정수 연산

| 함수 | 설명 | 가능 연산자 | 점수 | 연산자 개수 제한 |
|---|---|---|---|---|
| bitXor(x,y) | 두 변수의 XOR 연산 | ~ & | 1 | 4 |
| Tmin() | 2의 보수 정수형에서 가장 작은 수를 출력 | ! ~ & ^ \| + << >> | 1 | 4 |
| isTmax(x) | 2의 보수 표현에서 가장 큰 수인지 검사 | ! ~ & ^ \| + | 1 | 10 |
| allOddBits(x) | 홀수 번째 비트가 모두 1 인지 확인 | ! ~ & ^ \| + << >> | 2 | 12 |
| Negate(x) | 변수 x에 대해 -x를 반환 | ! ~ & ^ \| + << >> | 2 | 5 |

# 과제 내용 (bits.c)

- 정수 연산

| 함수 | 설명 | 가능 연산자 | 점수 | 연산자 개수 제한 |
|---|---|---|---|---|
| isAsciiDigit(x) | 변수 x가 ASCII코드 '0'(0x30)과 '9'(0x39)사이에 해당하는 문자인지 판별 | ! ~ & ^ \| + << >> | 3 | 15 |
| conditional(x, y, z) | C언어의 3항 연산자인 x ? y : z를 구현 | ! ~ & ^ \| + << >> | 3 | 16 |
| isLessOrEqual(x, y) | y가 x보다 크거나 같을 경우 1, 아닌 경우 0 반환 | ! ~ & ^ \| + << >> | 3 | 24 |
| logicalNeg(x) | 논리연산자 '!'를, '!'를 사용하지 않고 구현 | ~ & ^ \| + << >> | 4 | 12 |
| howManyBits(x) | x를 2의보수로 나타내는데 필요한 최소 비트의 개수를 반환 | ! ~ & ^ \| + << >> | 4 | 90 |

# 과제 내용 (bits.c)

- 부동소수점 연산

| 함수 | 설명 | 가능 연산자 | 점수 | 연산자 개수 제한 |
|---|---|---|---|---|
| floatFloat2Int(uf) | 부동소수점 uf의 정수부분 계산 | 모든 정수 연산<br>(if, while 등 포함) | 4 | 30 |

# 과제 해법 예시

- bitOr(x,y)
  - 두 변수 x와 y의 OR 연산
  - 가능 연산자: ~ &

```
int bitOr(int x, int y) {
    return ~(~x & ~y);
}
```

- anyEvenBit(x)
  - 짝수 번째 비트에 1이 하나라도 있는지 확인
  - 가능 연산자: ! ~ & ^ |+ << >>

```
int anyEvenBit(int x) {
  int m8 = 0x55;
  int m16 = m8 | m8 << 8;
  int m32 = m16 | m16 <<16;
  int evenx = x & m32;
  return !!evenx;
}
```

# 몇 가지 힌트

- 유용한 비트 연산
  - 뺄셈: x – n = x + (n의 2의 보수) = x + (~n + 1)
  - 비교: (x == y) == !(x ^ y)
  - 마스크 (mask)

    – x & 0 = 0, x & ~0 = x

    – x | 0 = x, x | ~0 = ~0
- 인자 값에 따라 계산 방법이 다른 경우
  - 조건문(if) 없이 처리할 수 있도록 적절한 마스크(mask) 값을 활용
  - 특히, 인자 값이 0이 되는 경우를 잘 고려해야 함.
- **<u>bits.c에서 각 세부 과제 함수의 설명을 숙지할 것.</u>**
  - 동작 정의와 예, 허용되는 연산자 등

# [참고] 테스트 함수

- **tests.c**
  - bit.c 에서 구현한 연산의 결과가 정확한 지 검증하는 코드
    –코딩 규칙에 제약 없이 구현한 함수
  - 각 문제의 의미와 요구하는 결과를 이해하기 위해 참고할 수 있음
    .

# 점수 평가 방법

- 정확성 점수
  - 각 문제(함수)에 할당된 배점
  - btest 프로그램으로 확인
- 성능 점수
  - 제한된 연산자 개수를 만족하면 2점 부여 (초과하면 점수 없음)
  - 보다 효율적인 프로그램 작성 연습
- 프로그래밍 제약 준수 여부
  - 허용되지 않은 코딩 포함 시 감점
    - 문제에 따라 -50% ~ -100%

# 과제 제출 주의 사항

- 컴파일 여부를 반드시 확인한 후에 제출 (make  명령 실행)
  - 제출한 bits.c 파일에서 컴파일 오류가 발생하는 경우에는 과제 점수가 부여되지 않음.

- 과제 점수 확인 프로그램 실행 권장
  - btest, dlc, driver.pl 프로그램을 실행

–본인 점수 미리 확인



# 과제 수행 –따라 하기

- GitLab 본인 프로젝트에서 파일 확인 후 압축 풀기
  - “datalab-handout.tar”
  - $ tar xvf datalab-handout.ar
- 과제 컴파일
  - $ make

```
next@archi-01:~$ tar xf datalab-handout.tar
next@archi-01:~$ cd datalab-handout/
next@archi-01:~/datalab-handout$ ls
bits.c  btest.c  decl.c  Driverhdrs.pm  driver.pl  ishow.c   README
bits.h  btest.h  dlc     Driverlib.pm   fshow.c    Makefile  tests.c
next@archi-01:~/datalab-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
gcc -O -Wall -m32 -o fshow fshow.c
gcc -O -Wall -m32 -o ishow ishow.c
next@archi-01:~/datalab-handout$
```

# 과제 수행 –따라 하기

- 정답 확인 (btest, 오답인 경우)

```
next@archi-01:~/datalab-handout$ ./btest
Score   Rating  Errors  Function
ERROR: Test bitXor(-2147483648[0x80000000],-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test allOddBits(-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test byteSwap(-2147483648[0x80000000],0[0x0],0[0x0]) failed...
...Gives 2[0x2]. Should be -2147483648[0x80000000]
ERROR: Test replaceByte(-2147483648[0x80000000],0[0x0],0[0x0]) failed...
...Gives 2[0x2]. Should be -2147483648[0x80000000]
ERROR: Test rotateLeft(-2147483648[0x80000000],0[0x0]) failed...
...Gives 2[0x2]. Should be -2147483648[0x80000000]
ERROR: Test isTmin(-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 1[0x1]
ERROR: Test fitsBits(-2147483648[0x80000000],1[0x1]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test sign(-2147483647[0x80000001]) failed...
...Gives 2[0x2]. Should be -1[0xffffffff]
ERROR: Test isPositive(-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test addOK(-2147483648[0x80000000],-2147483648[0x80000000]) failed...
...Gives 2[0x2]. Should be 0[0x0]
ERROR: Test float_f2i(0[0x0]) failed...
...Gives 2[0x2]. Should be 0[0x0]
Total points: 0/26
next@archi-01:~/datalab-handout$
```

# 과제 수행 –따라 하기

- 정답 확인 (btest, 정답인 경우)

```
next@archi-01:~/datalab-handout$ ./btest
Score   Rating  Errors  Function
 1      1       0       bitXor
 2      2       0       allOddBits
 2      2       0       byteSwap
 3      3       0       replaceByte
 3      3       0       rotateLeft
 1      1       0       isTmin
 2      2       0       fitsBits
 2      2       0       sign
 3      3       0       isPositive
 3      3       0       addOK
 4      4       0       float_f2i
Total points: 26/26
next@archi-01:~/datalab-handout$
```

- 프로그래밍 제약 확인 (dlc, 오류가 있는 경우)

```
next@archi-01:~/datalab-handout$ ./dlc bits.c
dlc:bits.c:176:bitXor: Illegal operator (==)
dlc:bits.c:177:bitXor: Illegal operator (^)
dlc:bits.c:178:bitXor: Illegal if
next@archi-01:~/datalab-handout$
```

# 과제 수행 –따라 하기

- 총 점수 확인 (driver.pl) – 정확성과 성능

```
next@archi-01:~/datalab-handout$ ./driver.pl
1. Running './dlc -z' to identify coding rules violations.

2. Compiling and running './btest -g' to determine correctness score.
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c

3. Running './dlc -Z' to identify operator count violations.

4. Compiling and running './btest -g -r 2' to determine performance score.
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c


5. Running './dlc -e' to get operator count of each function.

Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
1       1       0       2       7       bitXor
2       2       0       2       7       allOddBits
2       2       0       2       17      byteSwap
3       3       0       2       6       replaceByte
3       3       0       2       16      rotateLeft
1       1       0       2       5       isTmin
2       2       0       2       7       fitsBits
2       2       0       2       4       sign
3       3       0       2       4       isPositive
3       3       0       2       9       addOK
4       4       0       2       13      float_f2i

Score = 48/48 [26/26 Corr + 22/22 Perf] (95 total operators)
next@archi-01:~/datalab-handout$
```