
실습 1

Minsoo Ryu

**Operating Systems and Distributed Computing Lab.
Hanyang University**

msryu@hanyang.ac.kr

목차

□ 준비 작업 및 참고 자료

- **GCC 설치**
- **Linux 명령어**

□ 실습

- **Bubble Sort**
- **Stack using Array**
- **Stack using Linked List**
- **Fibonacci Series**
- **Matrix Transpose**

준비 작업 및 참고 자료

GCC 설치

□ 설치 명령어

- `$ sudo apt update`
- `$ sudo apt install build-essential`
- `$ sudo apt install build-essential --fix-missing` (실패시)

□ 보충 설명

- “**sudo**”는 리눅스에서 일시적으로 **superuser**(관리자) 권한을 빌리는 명령어 (보통 **superuser** 권한이 필요한 프로그램 설치시 사용)
- “**apt**”는 **Advanced Package Tool**로서 **Ubuntu**와 같은 **Debian** 계열 리눅스에서 프로그램 패키지를 설치하는 명령어
- “**build-essential**”은 **C**와 **C++** 프로그램 빌드에 필요한 **libc**, **gcc**, **g++**, **make** 등을 포함하는 패키지

Linux 명령어 (1/3)

□ 종료 및 부팅

- **\$ shutdown -h now**
 - 즉시 시스템을 종료
- **\$ shutdown -r now**
 - 즉시 시스템을 재부팅

□ 디렉토리

- **\$ pwd**
 - 현재 디렉토리 위치 보기
- **\$ ls**
 - 현재 디렉토리 내용 보기
 - “\$ ls -al”은 상세한 정보를 표시
- **\$ cd ~**
 - 홈 디렉토리로 이동
- **\$ cd /**
 - 루트 디렉토리로 이동

Linux 명령어 (2/3)

□ 파일 관리

- **\$ cp source_file target_file**
 - 파일 복사하기
- **\$ cp -r source_dir_name target_dir_name**
 - 디렉토리 내용을 모두 포함하여 복사
- **\$ rm -rf dir_name**
 - 현재 디렉토리의 모든 파일과 모든 하위 디렉토리 내용을 삭제
- **\$ mv source_file target_file**
 - 파일 이름 변경하기 (또는 위치 바꾸기)

□ 파일 전송

- **\$ scp -r /home/user cdsn@192.168.2.8:/home**
- “/home/user”를 “cdsn@192.168.2.8:/home” 디렉토리로 전송

□ 파일 압축

- **\$ tar cvfz target.tar.gz source_directory_name**
 - source_directory_name 디렉토리의 내용을 묶어서(tar) 압축(gzip)
- **\$ tar xvfz target.tar.gz**
 - 압축 풀기

Linux 명령어 (3/3)

□ 디렉토리 검색

- **\$ find . -name "파일명"**
 - 현재 디렉토리 및 하위 디렉토리에서 "파일명" 탐색
- **\$ find . -name 파일명 2>/dev/null**
 - (허가 거부된 디렉토리에 대한) 실패 메시지를 화면에 표시하지 않으면서 탐색
- **\$ grep "STR" ***
 - 현재 디렉토리의 모든 파일에서 문자열 "STR" 포함 여부를 검색
- **\$ grep -r "STR" ***
 - 하위 디렉토리를 포함한 모든 파일에서 문자열 검색

□ 네트워크 정보 확인

- **\$ hostname**
 - 네트워크에서 식별자로 사용되는 호스트 이름을 조회
 - "\$ hostname -I"은 IP 주소를 조회
- **\$ ifconfig**
 - 네트워크 설정 확인
- **\$ iwconfig**
 - 무선 네트워크 설정 확인

□ 원격 접속

- **\$ ssh msryu@rpi**
 - "rpi" 호스트에 "msryu" 사용자로 접속

Linux Manual

❑ \$ man something

- 리눅스 명령어 및 **API**에 대한 상세한 설명을 제공

❑ 인터넷 man page (다수의 사이트에서 동일한 내용 제공)

- <https://man7.org/linux/man-pages/man1/>
- <https://linux.die.net/man/>
- ...

실습

1. Bubble Sort (1)

□ 구현 목표

- **Call-by-reference**를 이용하여 두 개의 변수값을 바꾸는 **swapVariable()** 함수를 작성하고, 이를 이용하여 주어진 정수값들을 정렬하는 **bubble sort** 알고리즘을 구현한다

□ 구현 조건

- 아래의 2개 함수를 구현하여 프로그램에 포함시킨다

```
void swapVariable(int* a, int* b)
```

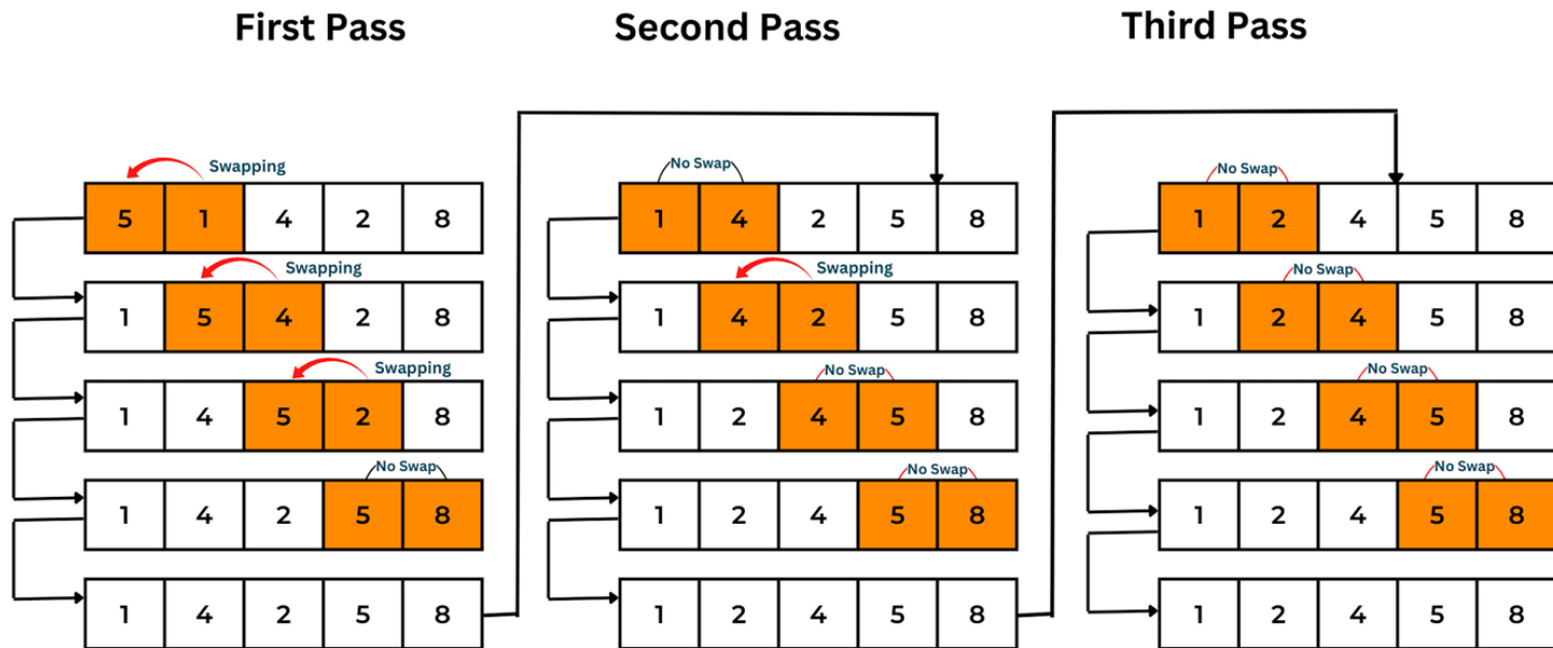
```
void bubbleSort(int dataTotal, int* dataArray)
```

- “**swapVariable**”는 **a**의 값과 **b**의 값을 맞바꾼다
- “**bubbleSort**”는 “**dataTotal**” 개수의 정수값을 담은 “**dataArray**” 배열을 받아서 정수값들을 증가하는 순서(**increasing order**)로 정렬한다

1. Bubble Sort (2)

□ 참고

- **Bubble sort** 알고리즘은 서로 인접한 두 원소의 대소를 비교하고, 조건에 맞지 않다면 자리를 교환하며 정렬하는 알고리즘



1. Bubble Sort (3)

□ 실행 및 결과 확인

- “input_sort.txt” 파일의 정수값들을 순서대로 읽어 정수형 배열에 저장한다
- **bubbleSort()** 함수를 호출하여 정수값들을 정렬한다
- 정렬된 결과를 **printf()** 함수로 화면에 출력한다

□ 실행 결과 예시

```
msryu@DESKTOP-AMQ05V8:~/Lab-Host/11_Bubble_Sort$ ./executable  
2 9 13 21 24 32 51 54 95
```

□ 제공될 자료

- “input_sort.txt” 파일
- “input_sort.txt” 파일을 읽어 드리는 “**int readInput(int* buffer)**” 함수의 소스 코드가 담긴 “**bubble.src**” 파일

2. Stack using Array (1)

□ 구현 목표

- 배열을 이용하여 정수형 데이터를 위한 스택 자료구조를 구현한다

□ 구현 조건

- 아래의 2개 함수를 구현하여 프로그램에 포함시킨다

`void push(int item)`

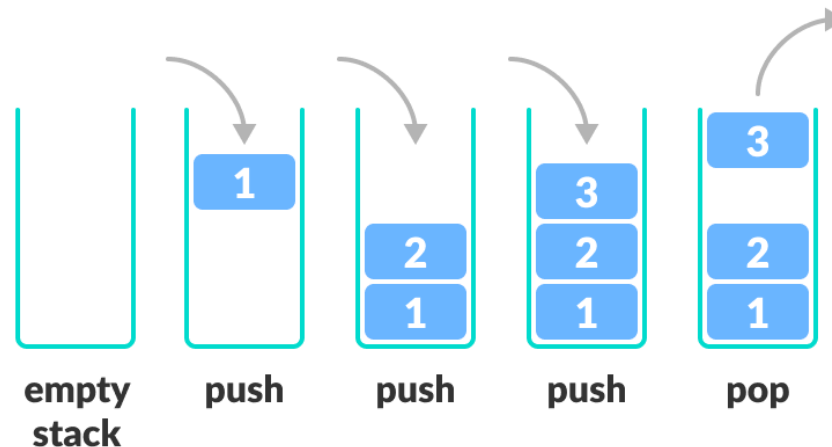
`int pop()`

- `push()` 함수는 `item`을 스택에 삽입한다
- `pop()` 함수는 스택의 `top` 위치에 있는 값을 리턴한다
- 제공될 “`input_array_stack.txt`” 파일에 포함된 입력 명령들을 순서대로 실행한 후 스택에 남아있는 값들을 `pop()` 순서로 화면에 출력한다

2. Stack using Array (2)

□ 참고

- 스택(**stack**)은 데이터 보관 및 관리를 위한 것으로 넣은 순서의 역순으로 데이터를 꺼내는 자료구조
- (주의) 지역변수를 저장하는 스택(**stack segment**)과 구별되어야 함



2. Stack using Array (3)

□ 실행 결과 예시

```
msryu@DESKTOP-AMQ05V8:~/Lab-Host/13_Stack_Array_Implementation$ ./executable  
The remaining elements in the stack: 50 40 10
```

3. Stack using Linked List (1)

□ 구현 목표

- 링크드 리스트를 이용하여 정수형 데이터를 위한 스택 자료구조를 구현한다

□ 구현 조건

- 아래의 구조체를 **push**와 **pop**의 단위로 사용한다

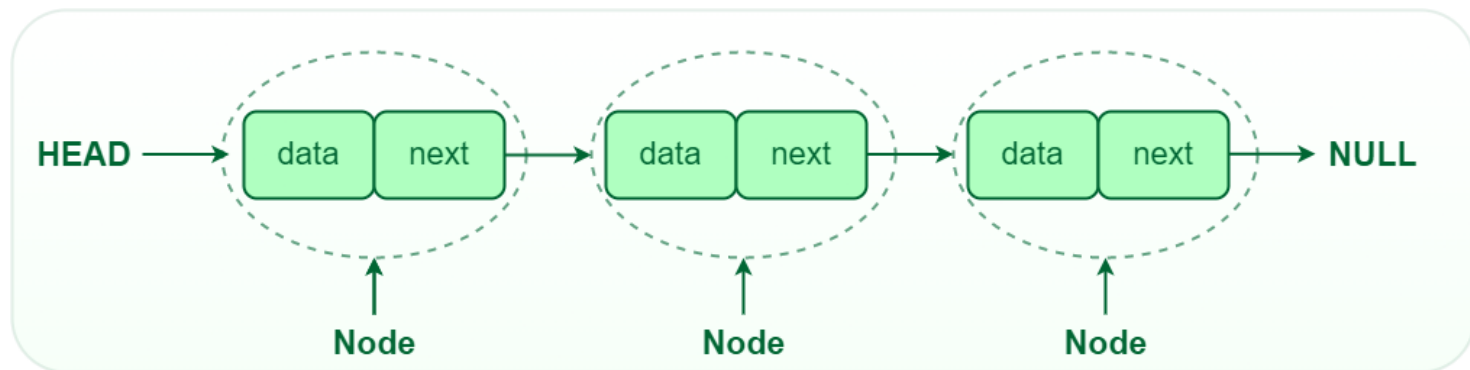
```
typedef struct Node {  
    int data;  
    struct Node* next;  
} Node;
```

- “**push()**” 함수에서는 **malloc()** 함수를 사용한다
- “**pop()**” 함수에서는 **free()** 함수를 사용한다
- 제공될 “**input_linked_list_stack**” 파일에 포함된 입력 명령들을 순서대로 실행한 후 스택에 남아있는 값들을 **pop()** 순서로 화면에 출력한다

3. Stack using Linked List (2)

□ 참고

- 링크드 리스트(linked list)는 포인터를 이용하여 데이터들을 연결하여 관리하는 자료구조



3. Stack using Linked List (3)

□ 실행 결과 예시

```
msryu@DESKTOP-AMQ05V8:~/Lab-Host/14_Stack_List_Implementation$ ./executable  
The remaining elements in the stack: 100 20 90
```

4. Matrix Transpose (1)

□ 구현 목표

- 다차원 배열을 이용하여 행렬을 표현하고 주어진 행렬을 전치행렬로 변환하는 프로그램을 작성한다

□ 구현 조건

- 아래의 함수를 구현하여 프로그램에 포함시킨다

`void swap(int* a, int* b)`


`int readInput(int inputData[][MAX_SIZE], int *rowSize, int *columnSize)`

`void generateMatrixTranspose(int rowSize, int columnSize, int matrix[][MAX_SIZE])`

4. Matrix Transpose (2)

□ 참고

- 전치행렬(transposed matrix)은 행렬 내의 원소를 대각선축(주대각성분)을 기준으로 서로 위치를 바꾼 것


$$A = \begin{matrix} (m \times n) \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \end{matrix}$$
$$A^T = \begin{matrix} (n \times m) \\ \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix} \end{matrix}$$

4. Matrix Transpose (3)

□ 실행 및 결과 확인

- “input_matrix.txt” 파일의 행렬을 “readInput()” 함수로 읽어 2차원 정수형 배열에 저장한다
- 읽어들이는 행렬을 화면에 출력한다
- “generateMatrixTranspose()” 함수를 호출하여 전치행렬로 변환한다
- 변환된 전치행렬을 화면에 출력한다

□ 제공될 자료

- “input_matrix.txt” 파일

4. Matrix Transpose (4)

□ 실행 결과 예시

```
msryu@DESKTOP-AMQ05V8:~/Lab-Host/16_Matrix_Transpose$ ./executable
```

11	12	13	14	15	16	17	18	19
21	22	23	24	25	26	27	28	29
31	32	33	34	35	36	37	38	39
41	42	43	44	45	46	47	48	49
51	52	53	54	55	56	57	58	59

11	21	31	41	51
12	22	32	42	52
13	23	33	43	53
14	24	34	44	54
15	25	35	45	55
16	26	36	46	56
17	27	37	47	57
18	28	38	48	58
19	29	39	49	59

5. Fibonacci Series (1)

□ 구현 목표

- 재귀호출(**recursive call**)을 이용하여 **Fibonacci** 수열을 계산하는 프로그램을 작성한다

□ 구현 조건

- 아래의 함수를 구현하여 프로그램에 포함시킨다

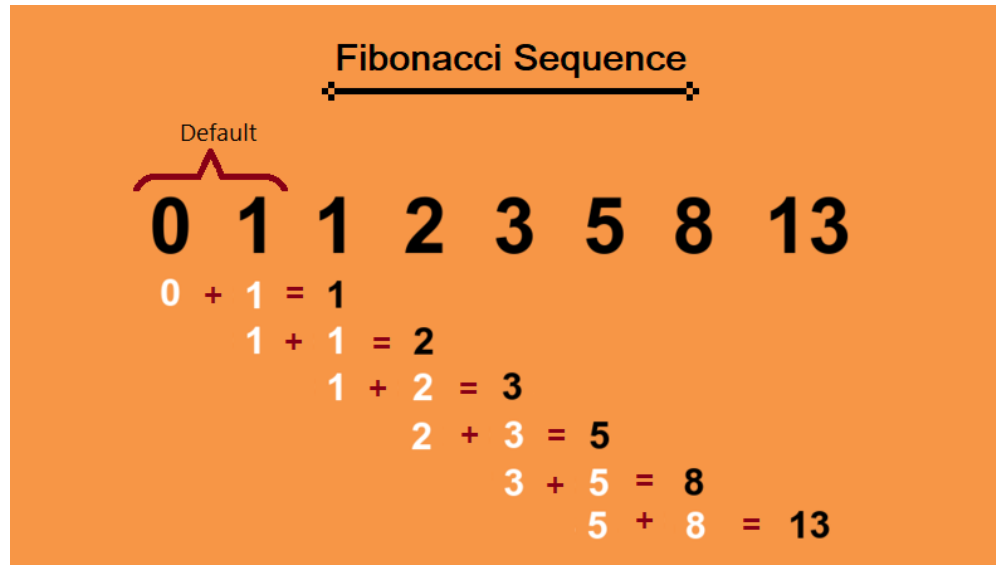
`long int fibonacci(int number)`

- **fibonacci()** 함수는 재귀호출을 사용한다
- 피보나치 수열의 **n**번째 항을 구할 때 **fibonacci()** 함수가 호출되는 총 횟수를 계산한다

5. Fibonacci Series (2)

□ 참고

- 피보나치 수열은 각 숫자가 이전 두 수의 합인 수열



5. Fibonacci Series (3)

□ 실행 및 결과 확인

- 사용자로부터 **n**의 값을 입력받는다(**scanf()** 함수 사용)
- 피보나치 수열의 **n**번째 항을 계산하여 화면에 출력한다
- 동시에 **Fibonacci** 함수가 호출된 횟수도 화면에 출력한다

□ 추가 구현

- 재귀호출 대신 루프(**loop**)를 사용하여 구현해보시오

5. Fibonacci Series (4)

□ 실행 결과 예시

```
msryu@DESKTOP-AMQ05V8:~/Lab-Host/15_Fibonacci$ ./executable
Enter a number up to which Fibonacci series are computed: 5

Fibonacci series computed by recursion.
Fib      Value      # of Calls
Fib(0)      0          1
Fib(1)      1          1
Fib(2)      1          1
Fib(3)      2          3
Fib(4)      3          5
Fib(5)      5          9

Fibonacci series computed by loop.
Fib      Value      # of Calls
Fib(0)      0          1
Fib(1)      1          1
Fib(2)      1          1
Fib(3)      2          1
Fib(4)      3          1
Fib(5)      5          1
```



thank you!