

System Programming and Practice Midterm Assignment

Student ID : 201721083

Name : 김태석

Q1.

코드 설명 : 코드는 크게 값을 입력 받는 부분, 입력 받은 값이 1,10,100의 자릿수 중 어디에 속하는지 판별하는 부분, 입력 받은 값과 150과의 대소 비교를 하는 부분, UP DOWN CORRECT 출력부분으로 나뉜다. 입력 받는 값은 최대 999까지 이므로 버퍼는 총 4바이트를 선언 해 주었다. 또한 각 자릿수를 T_1, T_10, T_100 변수에 저장하여 이를 각 자릿수에 1,10,100 곱한 값을 TEMP에 저장하였다.

알고리즘 : 버퍼 및 변수의 값을 0으로 초기화 시켜준 후 값을 입력 받는다. (입력 받는 값은 엔터 값을 기준으로 구분되며 최대 100의 자릿수 까지가 최대 범위로 한다.) 입력 받은 값의 자릿수를 판별하여 해당하는 자릿수 저장 섹션으로 이동한다. 해당 섹션에서 입력받은 바이트 값을 정수로 변환해준 뒤에 해당 값을 150과 비교한 후 알맞은 결과를 출력해주는 섹션으로 이동한다. 이 때 150과 값이 일치하면 CORRECT를 출력 후 프로그램을 종료시키고, 150 이상의 값일 경우 DOWN을 출력 후 다시 값을 입력 받는 섹션으로 돌아가고, 150 이하의 값일 경우 UP을 출력 후 다시 값을 입력 받는 섹션으로 돌아가서 위의 과정을 반복하여 실행 해 준다.

실행 결과

```
[kimtaeseok@MacBook-TaeSeok sictools % java -jar out/make/sictools.jar
7
UP
70
UP
120
UP
170
DOWN
500
DOWN
150
CORRECT
```

150 이하의 값을 입력 했을 경우 UP을 출력, 150 이상의 값을 입력 했을 경우 DOWN출력, 150과 같을 때 CORRECT를 출력한다. (1,10,100의 자릿수 모두 정상 작동하는 것을 확인 할 수 있다.)

에로사항 or 고찰 : 처음 코드를 짰을 때 150 이상, 이하 모두 프로그램이 종료되게 하여 다시 LOOP문으로 점프하여 코드를 실행시키자 RD로 바이트를 읽어 들였을 때 10(엔터값) 이 들어 있어 이후 실행 순서가 꼬이고 잘못된 값이 프린트 되는 일이 발생하였다. 해당 문제를 해결하기 위해 버퍼값을 초기화 해주는 부분을 추가하고 버퍼값에 아무런 값이 없을 경우 다시 LOOP문으로 돌아가는 부분을 추가 해 주어서 해당 문제를 해결 할 수 있었다.

Q2.

















코드 설명 : 코드는 노드의 개수를 입력 받는 부분, 노드들을 연결 해주는 정보를 입력 하는 부분, 공통 조상을 알고 싶은 쌍의 개수를 입력 받는 부분, 공통 조상을 알고 싶은 두개의 정점을 입력하는 부분, 공통 조상을 출력 해 주는 부분으로 나뉜다. 노드들을 연결 해주는 정보는 부모노드, 자식노드로 나뉘는데 부모노드는 STACK_L에, 자식 노드는 STACK_R에 저장 해 주도록 한다. 각 스택은 스택 속 값들을 서치할 수 있도록 두 스택 모두 POP PUSH 섹션을 따로 만들어 주었다.

알고리즘 : 스택을 사용하기 위해 처음에 스택을 선언해주고 스택 속 포인터를 설정 해 준다. 이후 노드의 개수를 입력하고 입력한 노드의 개수-1 개 만큼의 노드 연결 정보를 입력 받는다. 이 때 부모노드 자식노드 순으로 입력을 받는데 부모노드는 STACK_L, 자식노드는 STACK_R에 각각 저장하도록 한다. 저장시 PUSH를 이용하여 스택 포인터를 각각 이동시켜 다음 정보를 받아들여 저장할 수 있도록 한다. 또한 입력하는 노드의 정보를 자릿수별로 나누어 주어야 하므로 총 네가지의 경우를 나누어 버퍼에 들어온 값에 따라 해당 하는 정보를 적절한 섹션에서 취할 수 있도록 한다. 이후 공통 조상을 알고 싶은 쌍의 개수를 입력 받고, 입력 받은 개수만큼 공통 조상을 알고 싶은 두개의 노드를 입력해준다. 입력 받는 두개의 노드 또한 각 자릿수별로 적절히 경우를 나누어주어야 하는데, 노드 연결 정보를 입력 받을 때 사용한 섹션을 사용하여 코드를 간결하게 만들어 주었다. 공통 조상을 알고 싶은 두 노드를 입력하였다면 STACK_R에서 입력한 값을 찾아 같은 번째의 STACK_L에서 부모 노드를 찾아 두 값이 같다면 조상을 출력 한 후 다음 쌍을 입력 받고, 조상이 같지 않다면 다시 STACK_R에서 이전 단계에서 찾은 부모노드를 STACK_R에서 찾아 위의 과정을 반복하여 가장 가까운 공통 조상을 찾아준다.

실행 결과

```
[kimtaeseok@MacBook-TaeSeok sictools % java -jar out/make/sictools.jar
10
1 2
1 3
2 4
3 7
2 6
3 8
4 9
6 5
4 10
2
4 7
1
10 5
2
```

N1_P	008F0	2	000002
N2_P	008F3	2	000002
PARENT	0093E	2	000002

Name	Address	Decimal	Hex
✓  STACK_L	00856	1	000001
 STACK_L[1]	00859	1	000001
 STACK_L[2]	0085C	2	000002
 STACK_L[3]	0085F	3	000003
 STACK_L[4]	00862	2	000002
 STACK_L[5]	00865	3	000003
 STACK_L[6]	00868	4	000004
 STACK_L[7]	0086B	6	000006
✓  STACK_R	0088F	2	000002
 STACK_R[1]	00892	3	000003
 STACK_R[2]	00895	4	000004
 STACK_R[3]	00898	7	000007
 STACK_R[4]	0089B	6	000006
 STACK_R[5]	0089E	8	000008
 STACK_R[6]	008A1	9	000009
 STACK_R[7]	008A4	5	000005

각 스택에 입력한 값이 올바르게 들어간 것을 확인 할 수 있고 입력 과정의 마지막인 10과 5의 공통 조상을 찾은 결과 두 노드의 부모노드가 같아진 2노드가 가장 가까운 공통 조상임을 확인 할 수 있다.

예로사항 or 고찰 : LCA를 단순하게 찾는 방법은 노드의 깊이를 이용하는 방법인데, 두 정점을 둘의 깊이가 같아질 때 까지 계속하여 부모 노드로 이동하는 방식이다. 이 문제에서는 해당 부모 자식간의 연결에서 깊이에 대한 정보를 직관적으로 얻을 수 없어 스택 속에서 값을 찾고 그를 비교하며 같아질 때 까지 반복하는 방식을 사용하였다. 하지만 이 방법에는 치명적인 문제가 있는데, 만일 노드간 연결 정보를 부모노드 자식노드 순으로 주어지지 않고 랜덤이라면 뿌리노드가 1이라는 정보를 바탕으로 두 스택을 부모노드스택 자식노드스택으로 정리 해 주어야 하기 때문에 실행 결과가 나오는 속도가 현저히 줄어들 것이다. (부모노드스택 자식노드스택을 서로 나누는 방법은 자식노드스택에서는 같은 노드가 2번 이상 나올 수 없는 부분을 이용하는 것이다.)



Q3

코드 설명 : 코드는 6까지의 수를 카운트해주는 부분, 값을 순차적으로 더하여 변수에 저장해주는 부분, 값을 출력해주는 부분, 출력하는 값의 자릿수를 판별하는 부분으로 나뉜다. 첫째 및 둘째 항이 1이기 때문에 처음 시작시 이용 할 변수는 0과 1로 설정 해 주었다.

알고리즘 : 피보나치 수열은 첫째 항의 값이 0이고 두번째 항의 값이 1일 때 이후 항들은 이전의 두 항을 더한 값으로 이루어진 수열이다. 주어진 문제는 첫 번째 피보나치 수부터 여섯번째 피보나치 수를 출력하는 것이고 첫째 및 둘째 항의 값이 1이라고 주어졌기에 0번째 항의 값을 0이라 한 후 이후 항들의 값을 계산 해 준다. 계산한 값이 1의자리라면 MAKE_I, 10의자리라면 MAKE_II 섹션으로 가서 해당 정수를 바이트로 출력할 수 있도록 변환해 준 뒤 출력해 준 후 LOOP문으로 돌아 오게 된다. 여섯번째 피보나치 수까지 출력하기에 LOOP문에서 카운트가 6보다 크게 되면 프로그램을 종료한다.

실행 결과

```
kimtaeseok@MacBook-TaeSeok sictools % java -jar out/make/sictools.jar
1
1
2
3
5
8
13
```

 CUR	004A1	21	000015
 PRE	004A4	13	00000D

코드에서는 변수 CUR(현재 값)을 하나씩 출력해준 후 CUR(현재값)이 PRE(이전값)과 더해진 후 다음 값(TEMP)을 계산하고, 다음 반복을 위해 CUR값이 PRE값에 저장된 다음 CUR 변수에 TEMP값을 저장 해 주도록 구성되어 6번째 수 13이 PRE에 저장되었고 13+8=21값이 CUR에 저장되었다.

에로사항 or 고찰 : 실습 예제와 같이 링커를 이용하였다면 코드가 한결 깔끔해질 수 있었을 것 같다. 만일 실습 예제에서 배운 FACTORIAL을 피보나치 수열을 구성하는데에 응용한다면 main.asm의 COMP #10을 #6으로 바꾼 후 fact.asm에서 첫째항 둘째항이 1이도록 해준 후 MUL result 부분을 ADD result로 바꿔준다면 간단하게 피보나치 수열을 구현 할 수 있을 것이다.