0602

🡺0.c

노드의 삭제는 라이브러리가 아닌 사용자가 하도록 위임한다.

스택에 생성했으면 그냥 냅두면 되고, 힙에 생성했으면 사용자가 free해주면 된다.

🡺1.c

이전 자료구조는 노드의 생성 및 파괴를 사용자가 처리하므로 가볍다는 장점이있다.

하지만 listInitialize()에도 더미헤드가 여전히 동적할당을 하고있다는 단점이 있다. 이를 해결하기 위해 더미헤드를 스택에 생성해보자.

🡺2.c

🡺ppt26

이전 자료구조는 잘 동작하지만 역방향 탐색에 대한 비용이 높다는 단점이 있다. 이를 해결하기 위해 이중연결 리스트를 도입해보자.

🡺3.c

🡺ppt28, 29

listAdd() 와 listAddHead()함수는 둘다 노드를 추가하는 함수이지만 이원화 즉, 나뉘어있다는 단점이 있다. 즉, 이전 자료구조는 노드삽입정책이 이원화 되어있다는 단점이있다. 이를 해결하기 위해 삽입정책을 일원화 또는 일반화하도록 하자.

cf.이원화되면 나중에 유지보수할 때 건드려야할곳이 많아지며 버그가 발생할 가능성이 높아진다.

🡺4.c

이전 자료구조는 삽입 정책이 일반화되어있지만 즉,개발자한테만 맞춰져있지만 다른 사용자가 가져다 쓸 때 매개변수가 너무 복잡하고 가독성이 떨어진다는 것이다.

사용자가 사용하기 쉽도록 코드를 변경해보자.

🡺5.c

이전의 자료구조는 data에 int만 쓸수있다는 단점이있다. 모든 타입을 처리하기위해 void\*를 사용한 코드로 바꾸어보자

🡺6.c

🡺ppt32

void\* 는 모든 타입을 받을수 있지만 양날의 검이다. 프로세스가 죽을수도 있다.

그렇기에 구조체 안에 data있었는데 이것을 Person안에 node를 넣자

즉, 구조체안에 노드를 넣자!!

노드가 데이터를 담는 것이 아닌 데이터가 노드를 담는 것!

🡺7.c

🡺ppt33

🡺ppt34/35/36/37

🡺8.c

6.c의 내용과 7.c의 개념을 가지고.

또한 굉장히 복잡하므로 구조체안의 노드에 대한 정보를 매크로로 따로 빼자

🡺9.c

이전의 코드는 자료구조의 탐색이 어렵다. 이를 해결하기 위해 반복을 위한 함수를 제공한다.

즉, for문이 굉장히 복잡하다. 그래서 이를 메크로로 빼자.

Node head = {&head, &head}; 이부분 잘 보기!

🡺a.c

🡺ppt38

노드 삭제에 대한 함수

listRemove()와 어차피 Person만들 때 Node list도 같이 만들어지므로 굳이 null체크 할필요가 없다. 그러므로 체크문도 삭제.