**<자료구조 및 알고리즘>**

1. 프로그래밍언어

|  |  |
| --- | --- |
| 1세대 | 기계어, 어셈블리어, 동굴, 움집,  기계어 4+5, 4bit 0001 0100 0101 ->145  어셈블리어 add 4 5   1. 키보드로부터 4을 읽는다(코드 30~40줄) 2. 이 코드들이 주기억장치로 간다 3. 연산장치로 이동(load ax, 4 ->번역, 어셈블러) 4. load bx, 5 -> add |
| 2세대 | 자연어(고급언어): fortran(계산용), cobol(자료처리용) -초가집  -컴파일: 자연어를 기계어로 번역/의식의 흐름대로  -유지보수가 개발보다 비싸서 반성이 일어남(->구조적 프로그래밍 언어) |
| 3세대 | 구조적프로그래밍 언어 –양옥집, 배관, 전기, 하수도… ->설계도 만들자  -C언어(벨연구소에서 unix를 만들면서)  -고급언어는 하드웨어 못건들임(어셈블리어와 동시에 씀)  -그러나 C언어 안에 어셈블리를 끌고 들어옴(다른 언어와는 달리 하드웨어를 제어할 수 있다)  \*C언어  -모듈단위(함수, 프로시져)  -goto문을 없애자 |
| 4세대 | -부품용 프로그램을 만들자 ⇒ 객체지향! 63빌딩  \*객체지향  -미리 많이 부품을 만들어서 부품을 이어서 프로그램으로  -부품은 추상적이어야 한다  -은닉성  -상속  -다형성(형태가 다양하다)  부품: 부품을 묶음 ex.react, view등 |

1. 알고리즘, 자료구조

|  |  |
| --- | --- |
| 자료구조  (Data Structure) | \*대량 데이터를 효율적으로 관리할 수 있는 데이터 구조  \*효율적인 데이터 처리를 위해 데이터 특성에 따라 체계적으로 데이터를 구조화하는 것  \*데이터 사이의 관계를 반영한 저장구조 및 그 조작방법  \*데이터 특성에 맞는 자료구조 사용 |
| 알고리즘  (Algorithm) | \*어떤 문제를 풀기 위한 절차/방법  \*어떤 문제에 대해 특정한 ‘입력’을 넣으면 원하는 ‘출력’을 얻을 수 있도록 만드는 프로그래밍  \*알고리즘은 프로그래밍으로 되어 있음  \*컴퓨터가 알아들을 수 있는 비트로 표현 – 바이너리(이진파일)  \*어떠한 문제를 해결하기 위한 일련의 절차를 공식호한 형태  \*목적: 정확하고 효율적으로 결과값을 얻기 위해  \*[참고](https://librewiki.net/wiki/%EC%8B%9C%EB%A6%AC%EC%A6%88:%EC%88%98%ED%95%99%EC%9D%B8%EB%93%AF_%EA%B3%BC%ED%95%99%EC%95%84%EB%8B%8C_%EA%B3%B5%ED%95%99%EA%B0%99%EC%9D%80_%EC%BB%B4%ED%93%A8%ED%84%B0%EA%B3%BC%ED%95%99/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98_%EA%B8%B0%EC%B4%88)(LibreWiKi)  \*[참고2](https://visualgo.net/ko)(visualgo)  \*[참고3](https://www.geeksforgeeks.org/)(geeksforgeeks)  #중요개념   1. 시간복잡도 2. 자료구조 3. 정렬 |

-3세대에 출발, 생산성을 늘려보자, 과정과 절차(알고리즘)

-데이터측면에서 알고리즘: 자료구조

-1세대, 2세대: 결과중심 주의

-3세대: 과정(절차)중심 주의

-4세대: 결과중심주의(객체)

@문제푸는 곳

1. 백준
2. [프로그래머스](https://programmers.co.kr/learn/challenges)
3. [알고스팟](https://algospot.com/judge/problem/list/)
4. [코딩고장](http://codingdojang.com/)
5. [Hackerearth](https://www.hackerearth.com/practice/)
6. [Hackerrank](https://www.hackerrank.com/dashboard)
7. [Codeforces](https://codeforces.com/)
8. 자료구조

|  |  |
| --- | --- |
| 선형구조 | **①배열**  -프로그램 시작전에 메모리가 연속된 공간에 확보  -프로그램 수행 도중에 크기변경이 불가  -데이터를 중간에 끼워넣거나 중간에 삭제 불가  -고정된 위치에 고정된 크기를 차지한다  ------장점: 속도가 빠름  ------단점: 들어올 데이터크기만큼 항상 그 자리를 확보하고 있어야 함  -게임만들 때 많이 사용  -절대 힙공간에 할당되지 않는다(힙공간에 자리잡지 않는다)  \*파이썬은 모든 데이터가 힙에 들어감(본래적 의미의 관점이 깨진 것)  \*파이썬은 배열 없음  다른언어 같은 경우 int a; →스택  Int \*p=new int( ) →힙  a  p--------------------------------🡪int( )  \*스택: 시스템이 관리  \*힙: 개발자가 관리 |
| **②리스트**  -연결리스트: 미리 필요한 메모리 확보 없이 데이터가 들어오면 필요한 위치를 잡고 데이터를 저장하고 다음번 데이터의 위치값(여분의 메모리 필요)을 저장한다  -연결하기 위해 최소 2개의 인자가 필요하다  -데이터가 연달아 있을 필요가 없다  -유동적인 데이터일 경우 메모리가 절약된다  -데이터 건수가 100만개로 정해진 경우 위치값을 저장해야 하는 리스트보다 배열이 더 속도가 빠르고 전체 메모리도 배열이 더 안쓴다  **class Node:**  **data**  **next - 다음번 요소에 대한 주소값을 저장한다**  배열 – 시작과 끝이 어딘지 안다  리스트 – 시작과 끝에 대한 정보를 별도로 유지해야 한다  **class MyList:**  **head – 리스트의 시작**  **tail – 리스트의 끝**   1. 단일연결리스트는 다음번 요소, 순서대로 조회하기 때문에 역순조회가 힘들다 2. 이중링크드리스트는 내 앞의 요소와 내 다음의 순서를 조회하고 역순조회도 가능하다 |
| **③스택**  -먼저 들어간 데이터가 나중에 나오는 구조를 총칭하는 말  -들어오는 순서가 반대로 나오는 것  -함수와 밀접한 관계  -깊이 우선 탐색, 미로찾기  -트리순회(infix, prefix, postfix)  -인터럽트  -재귀호출  -미로찾기  -Last In Firtst Out(LIFO) 후입선출  \*왜 필요한지  -함수  main functionA functionB  call functionA( ) call functionB( )  **A’ B’**  **B’ push B**  **A’ push A’**  Push – 스택에 넣기  Pop – 스택으로부터 데이터 하나 제거  Underflow – 스택이 비었는데 만약 제거요청의 상황을 underflow  Overflow – 스택은 크기제한이 있는데, 스택의 크기를 벗어난 상황  isEmpty – 스택이 비었으면 True, 아니면 False  isFull – 스택이 꽉차면 True, 아니면 False |
| **④큐(Queue)**  -선입선출(first in first out) fifo  -버퍼  -메시지큐  -은행에서 줄 서기 |
| **⑤데큐** |
| 비선형구조 | **①트리**  \*트리는 루트가 있고 그것으로부터 파생되어서 노드가 만들어진다  \*트리는 부모자식개념이다(탐색기에서 바라보는 디렉토리 구조, 조직도)  \*html, xml도 트리구조 |
| **②그래프**  \*위로 올라갈 수 있는 경우(네트워크망)  \*트리의 차일드 노드에서 루트까지 갈 수 있으면, 그래프 구조가 된다  \*네이게이션 만들 때 알고리즘으로 많이 씀(최근에 가장 떠오르는 자료구조) |