1. 리스트, 스택, 큐
   1. 들어가며  
      자료구조: 프로그램의 동작 성능과 안정성 확보를 위해 알맞은 것을 선택해야함.  
      선형 자료 구조(Linear data structure)
   2. 연속된 자료구조와 연결된 자료구조  
      시간복잡도(Time Complexity): 특정 작업을 수행하는 데 걸리는 시간을 데이터 크기에 대한 수식으로 표현한 것.

|  |  |  |  |
| --- | --- | --- | --- |
| Data[0] | Data[1] | Data[2] | Data[i] |
| Base Address | B.A.+1\*sizeof(type) | B.A.+2\*sizeof(type) | B.A.+i\*sizeof(type) |

* + 1. 연속된 자료구조(Contiguous data structure)  
       모든 원소를 단일 메모리 청크에 저장함.  
       모든 원소에 바로 접근 가능->O(1)  
       중간에 원소를 삽입하는 것은 그 뒤를 전부 한번씩 미뤄야 함->O(n)  
       배열은 Static과 Dynamic으로 나눌 수 있음. Static은 Stack 메모리 영역에 저장되어 함수를 벗어날 때 자동으로 해제됨. Dynamic은 Heap 메모리 영역에 저장되며, 사용자가 해제하기 전까지 유지됨.  
       각 원소들은 연결되어 있기 때문에, 한 원소에 접근할 때 인근의 원소도 함께 Cache로 가져온다. 그로 인해 주변 원소에 접근하는 것이 빨라진다  
       (cache locality가 좋음).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| data | next | -> | data | next | -> | data | NULL |

* + 1. 연결된 자료구조(Linked data structure)  
       데이터를 노드라는 여러 개의 메모리 청크에 저장(서로 다른 메모리 위치).  
       Linked List: 각각의 노드가 data와 다음 노드를 가리키는 next를 가지고 있음. 맨 마지막 노드는 next에 NULL을 가짐.  
       특정 원소에 접근하기 위해선 Head 노드부터 차례대로 접근해야 한다->O(n)  
       대신 원소의 추가와 삭제는 Next만 바꿔주면 되므로 매우 빠름->O(1)
    2. C 스타일 배열의 제약 사항  
       메모리 할당과 해제를 직접 해야 한다. 하지 않으면 메모리 누수가 발생할 수 있음.  
       [] 연산자에서 배열 크기보다 큰 원소를 참조하는 것을 체크하지 못함. 잘못 했을 시, Segmentation Fault나 메모리 손상이 발생할 수 있음.  
       배열을 중첩했을 시, 코드 이해가 어려워짐.  
       깊은 복사(Deep Copy)가 기본으로 동작하지 않음.
  1. std::array  
     std::array<type, size> name으로 선언 가능.  
     []연산자와 at(index) 형태의 함수도 제공. at은 std::out\_of\_range 예외를 발생시킴. 그래서 []보다 조금 느리지만 예외처리를 편하게 할 수 있음.
     1. begin, end, front, back  
        begin: 첫번째 원소를 반환.  
        end: 마지막 원소를 반환.  
        front: 첫번째 원소의 참조를 반환.  
        back: 마지막 원소의 참조를 반환.