데이터 구조

강의 노트 3 목록

김유증, 박사 조교수
컴퓨터 과학 및 정보 공학부
한국 가톨릭 대학교, 대한민국

일반적인 데이

- 정수, 실수, 이중, 부울, 문자
  - 빌트인 타입, 기계 제공
  - 몇 가지 간단한 연산: 더하기, 곱하기, 읽기, 쓰기

- 일반적으로 프로그래머는 이러한 연산에 대한 지식 없이 구현

추상 데이터 유형(ADT)

- 추상 데이터 유형(ADT)은 데이터의 사양으로, 종종 연산 집합과 함께 사용됩니다.

- ADT는 추상적이므로 다음과 같습니다.
  - 구현과 무관하게
  - 구조화 도구를 사용하여 int, float, char로 구성된 기본 제공 유형을 기반으로 사용자 정의 데이터 유형을 정의합니다.

추상 데이터 유형(ADT)

- ADT에는 인터페이스가 있습니다.
  - 대중에게 공개
  - 생성, 삭제 등의 기본 작업 선언하기

- ADT는 세부 구현을 캡슐화합니다.
  - 데이터 및 데이터 속성 보호
  - 다양한 구현 수 사용
    - 사용자 프로그램에는 영향을 미치지 않습니다.
    - 언제든지 교체 및 업그레이드 가능

L은

- 엔티티 또는 요소의 주문 컬렉션
  - 머리와 꼬리가 있는 요소의 시퀀스입니다.
  - (C,Y,R)은 (Y,C,R)과 다릅니다.
  - 순서가 없는 세트와 다릅니다.

- 운영
  - 시작, 끝 또는 k번째 위치에 요소 추가하기
  - 시작, 끝 또는 k번째 위치에서 요소 삭제하기
  - 시작, 끝 또는 k번째 위치의 요소에 액세스합니다.
  - 요소 찾기

배열 L은

- 배열 목록: 목록은 배열로 구현할 수 있습니다.

- 목록의 최대 요소 수는 처음에 알고 있어야 합니다.
  - 확장성 문제: 초기 크기가 작으면 유용하지 않고, 초기 크기가 크면 유용합니다. 낭비가 될 수 있습니다.

```
1  #define MAX_LEN 100
2
3  int A[MAX_LEN], L=0;
4
5  // initalize
6  L = 10
7  for( i=0 ; i<L ; i++ ) A[i] = rand();
8
```

배열 L은

- 요소 추가(잘못된 구현)

```
#define MAX_LEN 100

int A[MAX_LEN], N=0;

// initalize
N = 10
for( i=0 ; i<N ; i++ ) A[i] = rand();

// add at the beginning
for( i=0 ; i<N ; i++ ) A[i+1] = A[i];
A[0] = rand();
N++;

// add at the end
A[N] = rand();
N++;

// add at the k-th position
k = 6
for( i=k ; i<N ; i++ ) A[i+1] = A[i];
A[k] = rand();
N++;
```

배열 L은

- 요소 추가(유효한 구현)

```
1  #define MAX_LEN 100
2
3  int A[MAX_LEN], N=0;
4
5  // initialize
6  N = 10;
7  for( i=0 ; i<N ; i++ ) A[i] = rand();
8
9  // add at the beginning
10 for( i=N ; i>0 ; i-- ) A[i] = A[i-1];
11 A[0] = rand();
12 N++;
13
14 // add at the end
15 A[N] = rand();
16 N++;
17
18 // add at the k-th position
19 k = 6;
20 for( i=N ; i>k ; i-- ) A[i] = A[i-1];
21 A[k] = rand();
22 N++;
```

배열 L은

- 요소 삭제하기

```
1  #define MAX_LEN 100
2
3  int A[MAX_LEN], N=0;
4
5  // initialize
6  N = 10;
7  for( i=0 ; i<N ; i++ ) A[i] = rand();
8
9  // delete at the beginning
10 for( i=0 ; i<N ; i++ ) A[i-1] = A[i];
11 N--;
12
13 // delete at the end
14 N--;
15
16 // delete at the k-th position
17 k = 6;
18 for( i=k+1 ; i<N ; i++ ) A[i-1] = A[i];
19 N--;
```

배열 L은

- 요소 액세스(읽기/쓰기)

```
1  #define MAX_LEN 100
2
3  int A[MAX_LEN], N=0;
4
5  // initalize
6  N = 10
7  for( i=0 ; i<N ; i++ ) A[i] = rand();
8
9  // read at k-th position
10 k = 6
11 A[k];
12
13 // write at k-th position
14 k=5
15 A[k] = rand()
```

배열 L은

- 요소 찾기
  - 최악의 경우 실행 시간은 얼마나 걸리나요?

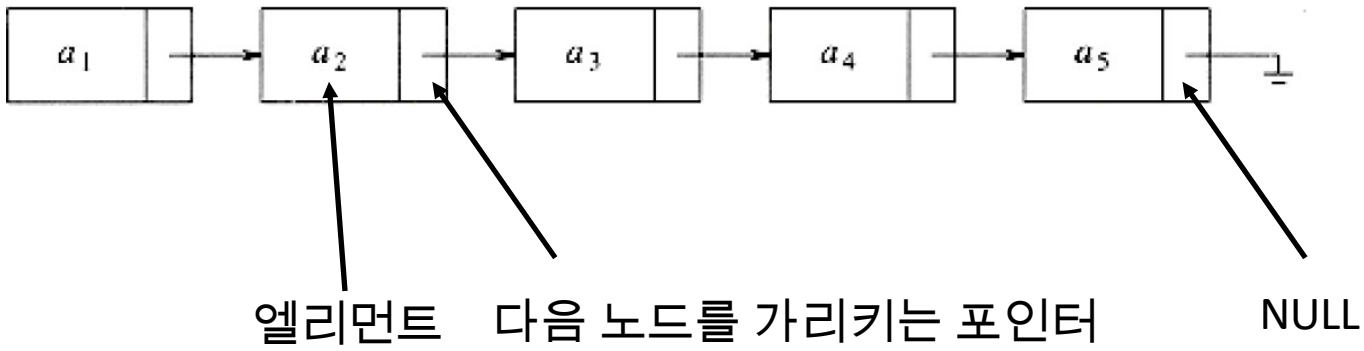
```
1  #define MAX_LEN 100
2
3  int A[MAX_LEN], N=0, toFind;
4  int* found;
5
6  // initalize
7  N = 10
8  for( i=0 ; i<N ; i++ ) A[i] = rand();
9
10 // find an element
11 toFind = 1234;
12 found = NULL;
13 for( i=0 ; i<N ; i++ ) {
14     if( A[i]==ToFind ) {
15         found = &A[i];
16         break;
17     }
18 }
19
20 printf("%d",(*found));
```

L은 잉크

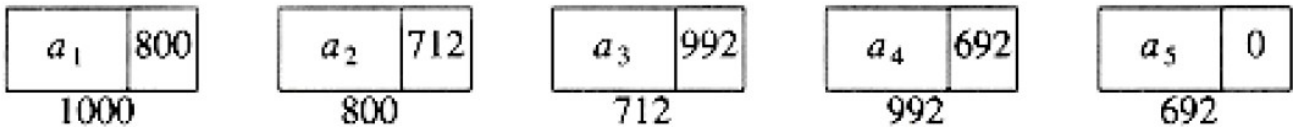
- 배열 목록은 단순하지만 확장성이 떨어지고 일부 연산은 효율적이지 않습니다(예: 처음에 추가하는 경우 O(N)).

- 연결된 목록
  - 메모리에서 인접한 위치를 사용하는 대신 각 노드에는 다음과 같은 요소가 포함됩니다.

    와 다음 노드에 대한 위치 포인터를 반환합니다.

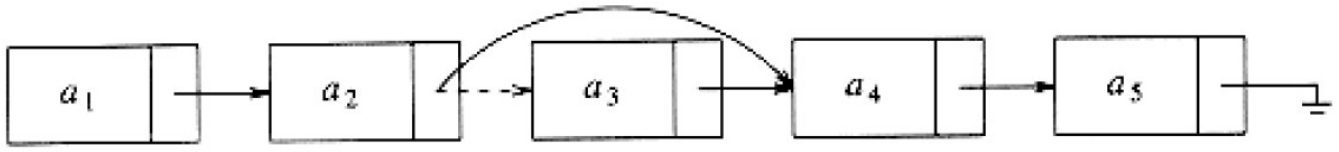


  - 실제 구현은 다음과 같을 수 있습니다:

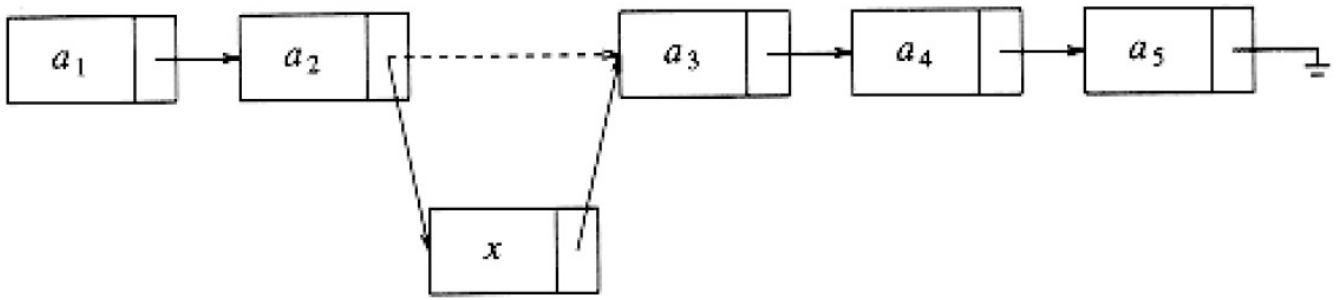


L은 잉크

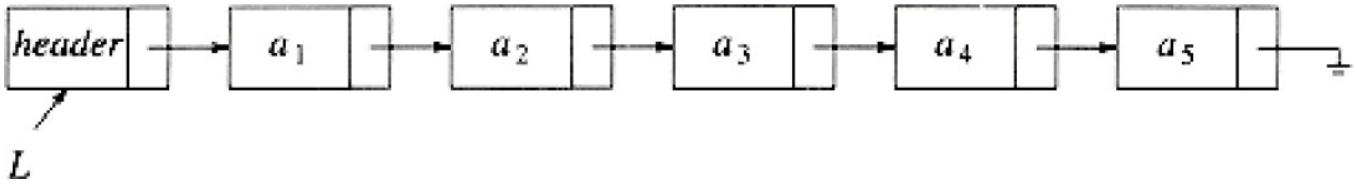
- 요소 추가(또는 삽입)



- 요소 삭제



- 실제 구현에서는 더미 헤드 노드를 사용하는 것이 편리합니다.



L은 L입니다: 구현

```
4  typedef int ElementType;
5
6  struct Node;
7  typedef struct Node *PtrToNode;
8  typedef PtrToNode List;
9  typedef PtrToNode Position;
10
11 List CreateList();
12 List MakeEmpty(List L);
13 int IsEmpty(List L);
14 int IsLast(List L);
15 Position Find(ElementType X,List L);
16 int Delete(ElementType X, List L);
17 Position FindPrevious(ElementType X, List L);
18 Position Insert(ElementType X, List L, Position P);
19 void DeleteList(List L);
20 Position Last(List L);
21 int ListLen(List L);
22 void PrintList(List L);
23
24 struct Node {
25     ElementType Element;
26     Position Next;
27 };
28
29 void FetalError(const char *msg)
30 {
31     printf(msg);
32     exit(-1);
33 }
```

L은 L입니다: 구현

```
35  int main(int argc, char *argv[])
36  {
37      int i;
38      ElementType X;
39      List L;
40      Position P;
41
42      // creat an empty list
43      printf("** Create an empty list\n");
44      L = CreateList();
45      PrintList(L);
46
47      // Insert at the beginning
48      printf("** insert 5 random numbers at the beginning\n");
49      for( i=0 ; i<5 ; i++ ) {
50          Insert(rand(),L,L);
51          PrintList(L);
52      }
53
54      // Insert at the end
55      printf("** insert a random number at the end\n");
56      P = Last(L);
57      Insert(rand(),L,P);
58      PrintList(L);
59
60      // Find
61      printf("** find the second element\n");
62      X = L->Next->Next->Element;
63      P = Find(X,L);
64      printf("%d is %s\n",X,(P)?"found":"not found");
65
66      // insertion in the middle
67      printf("** insert a random number in the middle\n");
68      Insert(rand(),L,P);
69      PrintList(L);
70
71      // Delete
72      printf("** delete %d from the list\n",X);
73      Delete(X,L);
74      PrintList(L);
75      printf("** find %d in the list\n",X);
76      P = Find(X,L);
77      printf("%d is %s\n",X,(P)?"found":"not found");
78
79      // Delete the whole list
80      printf("** delete the whole list\n",X);
81      DeleteList(L);
82      PrintList(L);
83
84      return 0;
85  }
```

```
C:₩Users₩jseok₩Documents₩[강의]₩2019 2 KECE231 데이터구조및알고리즘₩Sample Codes₩Li
** Create an empty list
0: NULL
** insert 5 random numbers at the beginning
1: 41 -> NULL
2: 18467 -> 41 -> NULL
3: 6334 -> 18467 -> 41 -> NULL
4: 26500 -> 6334 -> 18467 -> 41 -> NULL
5: 19169 -> 26500 -> 6334 -> 18467 -> 41 -> NULL
** insert a random number at the end
6: 19169 -> 26500 -> 6334 -> 18467 -> 41 -> 15724 -> NULL
** find the second element
26500 is found
** insert a random number in the middle
7: 19169 -> 26500 -> 11478 -> 6334 -> 18467 -> 41 -> 15724 -> NULL
** delete 26500 from the list
6: 19169 -> 11478 -> 6334 -> 18467 -> 41 -> 15724 -> NULL
** find 26500 in the list
26500 is not found
** delete the whole list
0: NULL
```

L은 L입니다: 구현

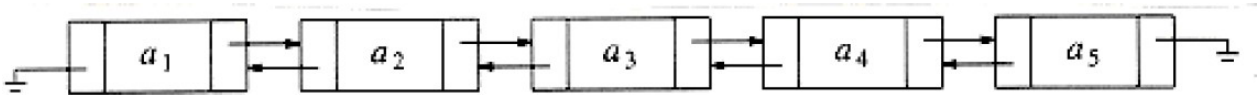
```
87  List CreateList()
88  {
89      List L;
90      L = (List)malloc(sizeof(struct Node));
91      if( L==NULL ) FetalError("Out of Memory");
92      L->Element = 0;
93      L->Next = NULL;
94      return L;
95  }
96
97  int IsEmpty(List L)
98  {
99      return( L->Next==NULL );
100 }
101
102 int IsLast(List P)
103 {
104     return( P->Next==NULL );
105 }
106
107 Position Find(ElementType X,List L)
108 {
109     Position P = L->Next;
110     while( P!=NULL && P->Element!=X) P = P->Next;
111     return P;
112 }
```

L은 L입니다: 구현

```
114  int Delete(ElementType X,List L)
115  {
116      Position P, Tmp;
117      P = FindPrevious(X,L);
118      if( IsLast(P) ) return 0;
119      Tmp = P->Next;
120      P->Next = Tmp->Next;
121      free(Tmp);
122      return 1;
123  }
124
125  Position FindPrevious(ElementType X,List L)
126  {
127      Position P = L;
128      while( P->Next!=NULL && P->Next->Element!=X)
129          P = P->Next;
130      return P;
131  }
132
133  Position Insert(ElementType X,List L,Position P)
134  {
135      Position Tmp;
136      Tmp = (Position)malloc(sizeof(struct Node));
137      if( Tmp==NULL ) FetalError("Out of Memory");
138      Tmp->Element = X;
139      Tmp->Next = P->Next;
140      P->Next = Tmp;
141      return Tmp;
142  }
```

다양한 L 잉크

- 이중 링크 목록: 다음 항목과 이전 항목에 대한 포인터가 있습니다.




```
27  struct Node {
28      ElementType Element;
29      Position Next;
30      Position Previous;
31  };
```

- 순환 링크 목록: 머리와 꼬리 없음