PSet listnode: a singly-linked list

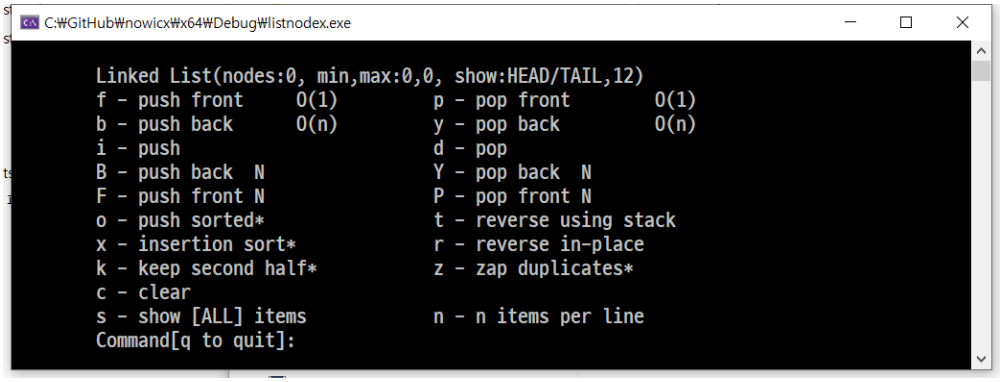
목차

Getting Started

이 PSet은 단일 연결 리스트의 노드로 구성되어 있습니다. 헤더 구조와 센티널 노드가 없는 단일 연결 리스트를 구현한 **listnode.cpp** 프로그램을 완성하세요. 노드와 노드를 단일 연결하면 첫 번째 노드가 항상 헤드 노드가 됩니다.

- listnode.cpp - 뼈대 코드, **대부분의 코드를 구현할 파일.**
  listnode.h - 인터페이스 파일, **수정 금지**
- driver.cpp - 구현한 코드를 테스트하기 위한 드라이버 파일
- listnodex.exe - pc용 실행 예시 파일

실행 예시:



Step 1: push_front(), push_back(), push()

첫 번째 함수 **push_front()**는 노드를 리스트 앞에 삽입합니다. 별도의 헤더 구조를 유지할 필요가 없으므로 첫 번째 노드가 항상 헤드 노드의 역할을 합니다. 새 노드가 맨 앞에 삽입되므로 이 함수는 헤드 노드를 가리키는 새 포인터를 호출자에게 반환해야 합니다. 호출자는 반드시 첫 번째 노드의 정보를 재설정해야 합니다. 이 연산의 시간 복잡도는 O(1)입니다. 이 함수는 이미 뼈대 코드에 구현되어 있습니다.

Push_back() 함수는 새 노드를 기존 노드 리스트의 끝에 추가합니다. 리스트가 존재하지 않으면 새 노드가 헤드 노드가 됩니다. 새 노드를 끝에 추가하려면 리스트의 노드를 모두 거쳐 마지막 노드를 찾아야 합니다. 마지막 노드를 찾은 후 새 노드를 추가하면 됩니다. 이 연산의 시간 복잡도는 O(n)입니다.

Push() 함수는 **x**를 가진 노드 위치에 **val**을 가진 새 노드를 삽입합니다. 사실상 새 노드는 **x를 가진 노드 앞에 삽입**됩니다. 이 함수는 리스트의 첫 번째 노드를 반환합니다. 컨테이너 크기가 1 증가합니다.

- 리스트가 비어 있으면 **val**을 가진 새 노드는 첫 번째 노드 또는 리스트의 헤드가 됩니다. 이 경우에 인수 x는 무시합니다.
- X를 가진 노드를 찾을 수 없으면 **val**을 가진 새 노드를 리스트 끝에 삽입합니다.
- 리스트에 존재하는 노드가 하나여서 그 자리에 새 노드를 삽입해야 하는 경우 주의하세요. 이 경우에 x를 이용해 노드의 위치를 찾을 수는 있지만, **val**을 가진 새 노드와 연결할 **prev** 노드가 존재하지 않습니다.

힌트: 새 노드를 x를 가진 노드 앞에 삽입하기 위해서는 x를 가진 노드의 이전 노드와 x를 가진 노드가 모두 있어야 합니다. **clear()**에서도 동일한 것이 사용되므로 **clear()**의 코드 예제를 참조하세요.

```
pNode push(pNode p, int val, int x);
```

중요 노트: 이 step에서 점수를 얻기 위해서는 강의에 나온 해당 파트의 코드를 간소화해야 합니다. 강의에서 사용된 함수의 이름은 “insert()”입니다. 강의 자료에서 해당 파트의 코드를 복사하여 사용하되 while() 루프에서 “if”문을 사용하지 않도록 간소화하세요. While()과 if()의 두 조건을 합쳐서 하나의 while() 루프로 간소화할 수 있습니다.

While() 루프 이후에 if문을 **사용하지 않아도** 해당 파트를 완성할 수 있습니다. 어떻게 작동하는지, 무엇을 반환해야 하는지, 이 두 경우를 생각해 보세요. pop()에서는 while() 루프 이후에 if문이 필요할 수 있습니다.

Step 2: pop_front(), pop_back(), pop()

이 함수들은 연결 리스트에서 노드를 삭제합니다. **pop_front()**는 리스트의 첫 번째 노드를 제거하고, 두 번째 노드가 첫 번째 노드 또는 헤드 노드가 됩니다. 이 함수는 두 번째 노드였던 새로운 첫 번째 노드를 가리키는 포인터를 반환합니다. 이 연산의 시간 복잡도는 O(1)입니다.

pop_back()은 리스트의 마지막 노드를 제거합니다. 마지막 노드를 제거하기 위해서는 리스트의 노드를 모두 거쳐 **마지막 노드**와 마지막 노드의 **이전 노드**를 찾아야 합니다. 마지막 노드가 제거된 후 그 이전 노드가 마지막 노드가 되므로 다음 필드를 **nullptr**로 설정하기 위해 마지막 노드의 이전 노드도 필요합니다.

pop() 은 사용자가 지정한 특정 값을 가진 노드를 제거합니다. 리스트에 존재하는 임의의 노드를 제거할 수 있습니다.

중요 노트: 이 step에서 점수를 얻기 위해서는 강의에 나온 해당 파트의 코드를 간소화해야 합니다. 강의 자료에서 해당 파트의 코드를 복사하여 사용하되 while() 루프에서 “if”문을 사용하지 않도록 간소화하세요. While()과 if()의 두 조건을 합쳐서 하나의 while() 루프로 간소화할 수 있습니다.
이 파트를 완성하기 위해서 while() 루프 이후에 “if”문이 필요할 수 있습니다.

Step 3: show(), clear(), minmax() - liststack.cpp에서 복사해오세요.

show()는 스택 또는 연결 리스트의 노드를 위에서부터 아래로 출력합니다. 이 함수에는 **bool show_all = true**라는 선택적 인수가 있습니다. 드라이버의 메뉴 옵션을 통해 사용자는 “show [ALL]”과 “show [HEAD/TAIL]”을 전환할 수 있습니다. 이 기능은 코드 디버깅에 유용합니다.

이 함수에는 또 다른 선택적 인수인 **int show_n = 10**이 있습니다. 리스트의 크기가 show_n * 2보다 작거나 같으면 리스트의 모든 항목들을 출력합니다. “show [ALL]” 옵션에서 “show [HEAD/TAIL]” 옵션으로 전환되고, “show [HEAD/TAIL]” 옵션에서 “show [ALL]” 옵션으로 전환됩니다. [HEAD/TAIL] 옵션은 최대 show_n개의 아이템을 리스트의 처음부터 끝까지 출력합니다.

clear() 함수는 리스트의 모든 노드의 할당을 해제합니다. “delete”를 N번 출력해야 하며, N은 노드의 개수입니다. x를 가진 노드를 제거하기 위해서 x를 가진 노드의 이전 노드와 x를 가진 노드 자체를 모두 가지고 있어야 합니다. **clear()**에서도 동일한 것이 사용되므로 **clear()**의 코드 예제를 참조하세요. 이 함수는 이미 구현되어 있습니다. It is a grace~.

minmax() 함수는 리스트에서 최솟값과 최댓값을 찾습니다. 스택 ADT가 최솟값과 최댓값을 가질 필요는 없지만, 코드의 무결성을 확인하기 위한 작업입니다.

```
// sets the min and max values which are passed as references in the list
void minmax(pNode p, int& min, int& max);
```

Step 4: cut_in_two_halves()

이 함수나 알고리즘은 강의에서 설명하지 않았습니다. 강의에서 다룬 **keep_scond_half()**라는 유사한 함수를 참조하세요.

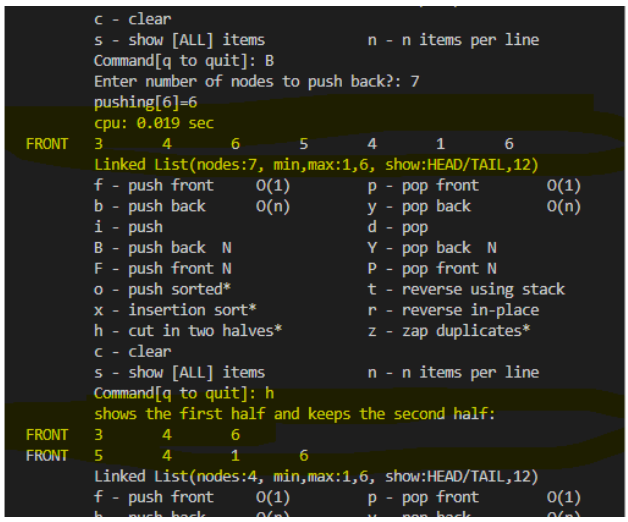
The function **cut_in_two_halves()** 함수는 리스트를 절반으로 나누고, 첫 번째 절반의 첫 번째 노드와 두 번째 절반의 첫 번째 노드를 반환합니다. 노드의 개수가 홀수라면 두 번째 절반이 노드를 하나 더 가져가도록 합니다. 예를 들어, 9개의 노드가 있다면 두 번째 절반은 5개의 노드를 가져갑니다. 리스트의 크기는 2보다 크거나 같아야 합니다.

중간 노드의 주소는 이전 노드의 next에 저장된다는 걸 기억하세요. 첫 번째 절반의 마지막 노드에서 nullptr 설정하는 것을 잊지 마세요.

```
pair<Node*, Node*> cut_in_two_halves(Node* p) {
   int n = size(p) / 2;
   int count = 0;

// your code here
```

```
  return make_pair.......;
}
```

실행 예시:



Step 5: "B", "F" and "Y", "P"

이 옵션들은 사용자에게 추가 또는 제거할 노드 수를 지정하고 작업을 수행하도록 요청합니다. 예를 들어, B와 F 옵션은 함수 하나를 제외하고 동일하게 작동합니다. “B” 옵션은 push_back() 함수를, “F” 옵션은 push_front() 함수를 N번 호출합니다. Y 옵션과 P 옵션도 위와 비슷합니다.

- "B" - 난수로 push_back()을 N번 호출합니다.
- "F" - 난수로 push_front()을 N번 호출합니다.
- “Y" - pop_back()을 N번 호출합니다.
- “P" - pop_front()를 N번 호출합니다.

“Y”와 “P” 옵션에서 사용자가 범위를 벗어나는 숫자를 지정하면 모든 노드를 제거합니다.

함수 포인터를 사용하여 두 가지 옵션을 수행하는 함수를 구현합니다. 사용자(혹은 드라이버)가 아래와 같이 선택 가능한 함수 포인터(push_front 혹은 push_back)을 전달하도록 합니다.

```
    case 'B':
      val = GetInt("\tEnter number of nodes to push back?: ");
      begin = clock();
      head = push_N(head, val, push_back);
      break;

    case 'F':
      val = GetInt("\tEnter number of nodes to push front?: ");
      begin = clock();
      head = push_N(head, val, push_front);
      break;
```

push_N()이 함수 포인터를 인수로 받은 후, 아래와 같이 함수 포인터를 사용하여 push_front()와 push_back() 중에 필요한 함수를 호출합니다:

```
// adds N number of new nodes at the front or back of the list.
```

```
// Values of new nodes are randomly generated in the range of
// [0..(N + size(p))].
// push_fp should be either a function pointer to push_front() or push_back().
Node* push_N(Node* p, int N, Node* (*push_fp)(Node *, int)) {

   // your code here
   push_fp(p, ...);
   ...
}
```

난수 생성을 위해 rand() 사용 시 참고 사항

일반적으로 RAND_MAX가 32767로 정의되어 있어서 rand()를 사용하여 난수를 생성하면 우리가 원하는 것보다 훨씬 작은 수가 생성됩니다. 뼈대 코드에 제공된 도우미 함수 rand_extended()를 활용하세요. 이 함수는 nowic/lib/librand.a, librand_mac.a 또는 rand.lib. 그리고 nowic/include/rand.h에서 확인할 수 있습니다.

```
// returns an extended random number of which the range_max is from 0
// to (RAND_MAX + 1)^2 - 1. // We do this since rand() returns too
// small range [0..RAND_MAX) where RAND_MAX is usually defined as
// 32767 in cstdlib. Refer to the following link for details
// https://stackoverflow.com/questions/9775313/extend-rand-max-range

unsigned long rand_extended() {
   if (range_max < RAND_MAX) return rand();
   return rand() * RAND_MAX + rand();
}
```

Step 6: 단일 연결 리스트 뒤집기 - reverse_using_stack()

스택을 이용해서 단일 연결 리스트를 뒤집습니다. 이 알고리즘은 제공된 강의 자료와 **강의에 자세히 설명되어 있습니다**.

입력된 리스트의 마지막 노드가 새로 형성된 리스트의 헤드 노드가 됩니다. 이 알고리즘은 아래와 같습니다.

1. 리스트의 모든 노드를 스택에 push( )합니다.
2. 스택에서 노드를 한 번에 하나씩 top( )한 후 다시 연결합니다.
3. 리스트의 새 헤드를 반환합니다.

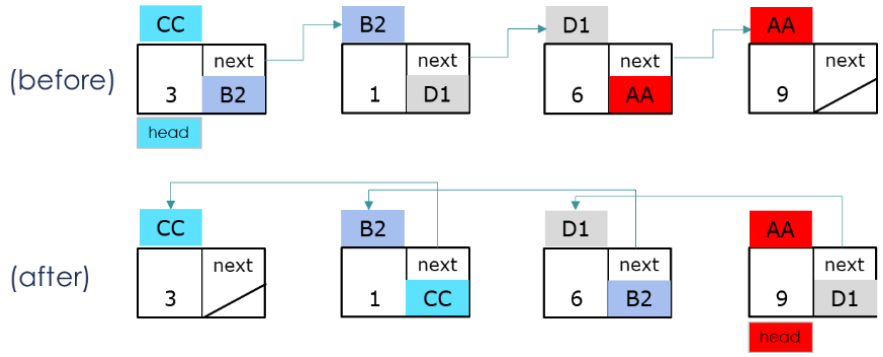
리스트의 모든 노드를 두 번 거쳐 가지만 시간 복잡도는 여전히 O(n)입니다. 하지만 이 알고리즘은 시간 복잡도가 똑같이 O(n)인 in-place reverse 알고리즘보다 훨씬 오래 걸립니다.

중요 노트: 노드를 재생성 하지 마세요. 노드를 먼저 스택에 push()한 후, 노드를 하나씩 다시 연결하면서 가져오면 됩니다. push_back(), push_front() 또는 new Node()와 같은 함수를 호출하지 마세요.

Step 7: 단일 연결 리스트 뒤집기 - reverse_in_place()

단일 연결 리스트를 스택이나 추가 메모리가 필요하지 않도록 제자리에서 뒤집습니다. 이 함수는 단일 연결 리스트를 뒤집고 새로운 헤드를 반환합니다. 입력된 리스트의 마지막 노드가 새로 형성된 리스트의 헤드 노드가 됩니다.

이 함수는 reverse_using_stack()보다 최소 2배 정도 빠르게 실행될 것입니다.



팁 & 힌트:

리스트의 모든 노드를 거쳐가며 두 노드의 포인터를 서로 교환하는 과정에서 할당(assignment)하기 전에 몇 개의 포인터를 저장해야 합니다.

while() 루프 이전에 prev = nullptr, curr = head로 설정하세요. while() 루프 안에서,

(1) curr->next에 새 포인터를 할당하기 전에 curr->next를 임의의 노드에 저장하세요.

(2) 루프에서 다음 노드로 넘어가기 전에 두 가지 사항을 확인하세요.

A. prev를 curr로 설정합니다. (예. curr이 prev가 됩니다)

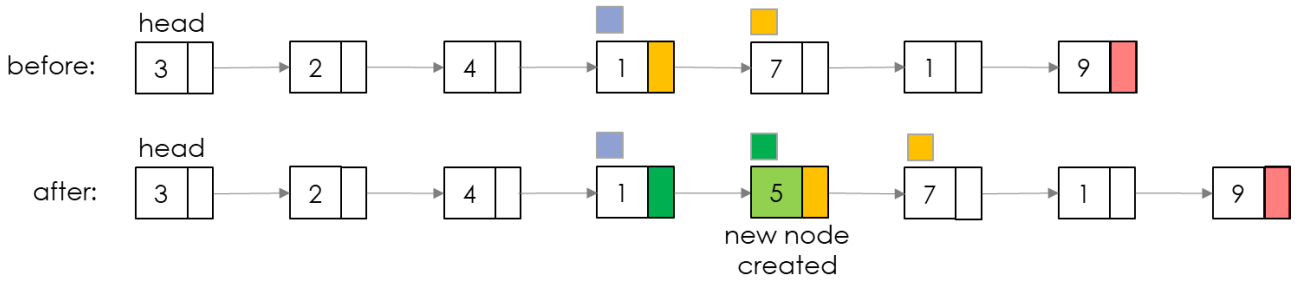
B. curr을 처리할 다음 노드로 설정합니다.

Step 8: push_sorted()*

push_sorted() 함수는 새 노드를 정렬된 오름차순으로 삽입합니다. 기본 전략은 새 노드를 삽입할 위치를 찾아 리스트의 노드를 반복(iterate)하여 거쳐가는 것입니다. 위치는 리스트의 끝일 수도, 노드의 바로 이전 위치일 수도 있습니다.

```
void push_sorted(Node* p, int value)
```

새 노드를 삽입할 위치(이 예시에서는 7)를 찾을 때, 주소가 이전 노드(이 예시에서는 1)에 저장되어 있는지 확인해야 합니다. 생성된 새 노드는 7의 주소를 갖게 되고, 이전 노드의 next는 새 노드의 주소로 업데이트해야 합니다.



Step 9: insertion_sort()*

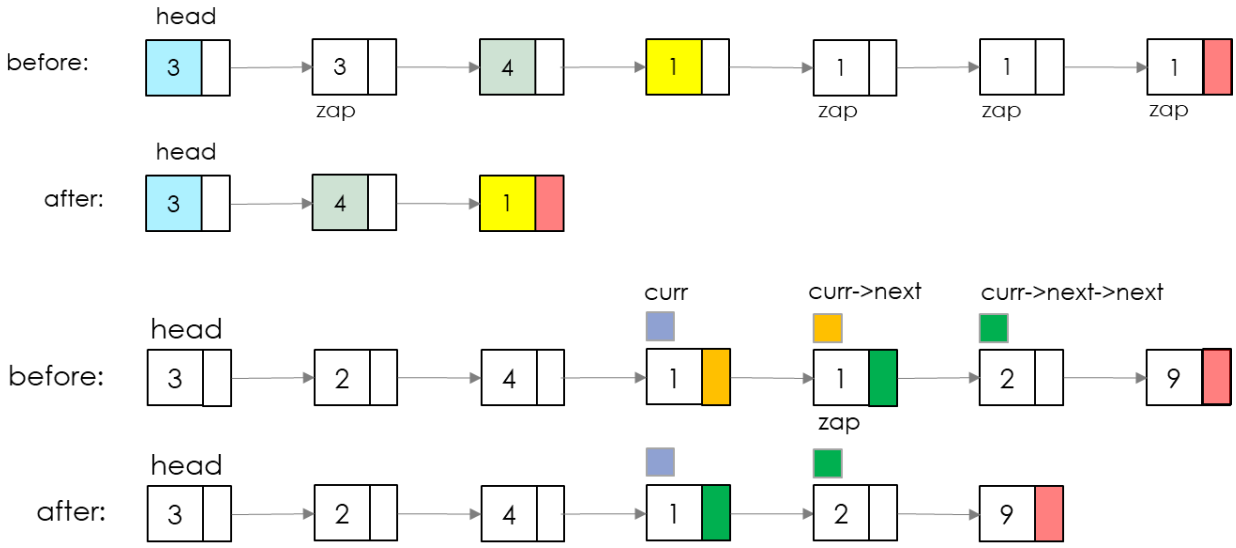
이 함수는 삽입 정렬을 사용하여 단일 연결 리스트를 정렬한 후, 정렬된 새 리스트를 반환합니다. push_sorted()가 새로 형성된 리스트의 헤드를 반환하도록 리스트에 있는 값을 사용하여 push_sorted()를 반복해서 호출하세요.

```
// inserts a new node with value in sorted order
Node* insertion_sort(Node* p) {
  if (empty(p)) return nullptr;
  if (size(p) < 2) return p;

  Node* sorted = nullptr;  // new list which will be formed using push_sorted()

  // your code here - a while loop invokes push_sorted() for all nodes in p

  clear(p);
  return sorted;
}
```

Step 10: zap_duplicates()

이 함수는 리스트에 존재하는 연속 항목들을 제거하고 인접 항목들을 고유하게 유지합니다. 리스트의 노드를 아래로 쭉 거쳐가면서 인접 노드와 비교합니다. 인접 노드와 동일한 경우, 두 번째 노드를 제거합니다. 한 가지 까다로운 부분은 노드를 제거하기 전에 다음 노드의 그다음 노드를 저장해야 한다는 것입니다.

리스트의 노드는 **오직 한 번**만 거쳐가야 합니다.



Step 11: 시간 복잡도

Big O 표기법을 사용하여 이 PSet에서 구현한 다음 작업들의 시간 복잡도를 완성하세요. **완성한 표를 스크린 캡처하여 게시하세요.**

작업	시간 복잡도	작업	시간 복잡도
f - push front	O(1)	p - pop front	O(1)
b - push back	O(n)	y - pop back	O(n)
i - push		d - pop	
B - push back N		Y - pop back N	
F - push front N		P - pop front N	
o - push sorted		t - reverse in-stack	
x - insertion sort		r - reverse in-place	

h - cut in two halves		z - zap duplicates	

과제 제출

- 소스 파일 상단에 아래와 같이 아너 코드 문장을 적고 서명하세요.
  On my honor, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  서명: _____ 분반: _____ 학번: _____
- 제출하기 전에 코드가 제대로 컴파일이 되고 실행되는지 확인하세요. 제출 직전에 급하게 코드를 수정한 후 코드가 제대로 컴파일이 될 거라고 짐작하지 않는 게 좋습니다. “거의” 작동하는 코드도 틀린 것입니다.
- 과제가 컴파일 및 실행된다면, 마감 기한 전까지 과제의 일부만 완성했더라도 제출하기 바랍니다. 컴파일 및 실행되지 않는다면 제출하지 마세요. 마감 시간 이후 24 시간 이내 제출하면, 만점에서 25% 감점하고 채점합니다. 그 이상 늦은 것은 채점하지 않으며, 0 점 처리합니다.
- 제출 후, **마감 기한 전까지** 수정 및 재제출이 가능합니다. 파일 하나만 수정하더라도 해당 파일과 관련된 파일들을 모두 재제출해야 합니다. 재제출 횟수는 제한 없습니다. 마감 기한 전에 **가장 마지막으로** 제출된 파일을 채점할 것입니다.

제출 파일 목록

아래에 명시된 파일들을 Piazza의 pset 폴더에 제출하세요.

- listnode.cpp - mid2 폴더 (종합 - 모든 Step을 포함함)

마감 기한 & 배점

- 마감일: 11:55 pm
- 총15점 - 각 Step 당 점수가 다를 수 있습니다.