Zadanie 2

Napisz fragmenty kodu wykonujące pewne działania na liście. Nie muszą być nawet w formie pliku, wystarczy wpisać je w formularzu do zadania. Chodzi o same fragmenty, **nie trzeba** wysyłać kompletnego kodu który da się skompilować. Fragmenty muszą być w konkretnym języku programowania, a nie w pseudokodzie.

1. **Dołączanie elementu na początek listy dwukierunkowej**
   Cel: Dołączanie elementu do listy dwukierunkowej na początek;
   Dane wejściowe:
   Położenie pierwszego elementu listy (np. wskaźnik na ten element);
   Dołączany element;
   Algorytm:

```
   List-Insert-Begin(L,x)
1: next[x]:= head[L];
2: if (head[L]!= NIL) then
3:     prev[head[L]] := x;
4: end if
5: head[L] := x;
6: prev[x] := NIL;

void push_front(List* L, T w)
{
  // Napisz definicję !!!
}
```

2. **Dołączanie elementu na koniec listy dwukierunkowej**
   Cel: Dołączanie elementu do listy dwukierunkowej na koniec;
   Dane wejściowe:
   Położenie ostatniego elementu listy (np. wskaźnik na ten element);
   Dołączany element;
   Algorytm:

```
   List-Insert-End(L,x)
1: if (tail[L]!= NIL) then
2:     next[tail[L]] := x;
3: end if
4: prev[x] := tail[L];
5: tail[L] := x;
6: next[x] := NIL;

void push_back(List* L, T w)
{
  // Napisz definicje!!!
}
```

3. **Usuwanie wskazanego elementu z listy jednokierunkowej**

Cel: Usunięcie wskazanego elementu z listy jednokierunkowej;

Dane wejściowe:

Położenie pierwszego elementu listy (np. wskaźnik na ten element);

Element do usunięcia;

Algorytm:

```
   List-Delete-1(L,x)
1: if (x==head[L]) then
2:     head[L] := next[x];
3: else
4:     y := head[L];
5:     while (next[y]!=x) do
6:         y := next[y];
7:    end while
8:     next[y]:= next[x];
9: end if
```

```
void pop_1(List* L, T w)
{
 // Napisz definicje!!!
}
```

4. **Usunięcie ostatniego elementu z listy dwukierunkowej**

(w tym podpunkcie nie ma pseudokodu)

```
void pop_back2(List* L)
{
 // Napisz definicje!!!
}
```

Przykłady pomocnicze:

1. **Dołączanie elementu na początek listy jednokierunkowej**

```
void push_front(List* L, T w)
{
  Node* p = malloc(sizeof(Node));
  p->value = w;
  p->next = NULL;
  if (L->first == NULL) { // lista pusta
    L->first = L->last = p;
  } else {
    p->next = L->first;
    L->first = p;
  }
}
```

2. **Dołączanie elementu na koniec listy jednokierunkowej**

```
void push_back(List* L, T w)
{
  Node* p = malloc(sizeof(Node));
  p->value = w;
  p->next = NULL;
  if (L->first == NULL) { /* lista pusta */
    L->first = L->last = p;
  } else { /* cos jest w liscie */
    L->last->next = p;
    L->last = p;
  }
}
```

3. **Usuwanie pierwszego elementu z listy jednokierunkowej**

```
void pop_front(List* L)
{
  // jeden element lub lista pusta
  if (L->first == L->last) {
    free(L->first);
    L->first = L->last = NULL;
  } else {
    Node* p = L->first;
    L->first = p->next;
    free(p);
  }
}
```

4. **Usuwanie ostatniego element z listy jednokierunkowej**

```
void pop_back(List* L) {
  // jeden element lub lista pusta
  if (L->first == L->last) {
    free(L->first);
    L->first = L->last = NULL;
  } else {
    Node* p = L->first;
    // szukamy przedostatniego elementu listy
    while ((p->next) != L->last) p=p->next;
    L->last = p;
    free(p->next); // i usuwamy go
    p->next = NULL;
  }
}
```