# Podstawy Programowania
# Zajęcia laboratoryjne 11
# I rok Bioinformatyki Politechniki Poznańskiej

Lista jednokierunkowa

## 1 Lista jednokierunkowa

**Lista** jest dynamiczną strukturą danych, dokładnie jest liniowo uporządkowaną strukturą zawierającą zbiór elementów, z których dowolny element można usunąć oraz dodać w dowolnym miejscu. Lista ma postać "kontenerową", gdzie każdy kontener (węzeł) zawiera pewne dane i wskaźnik na następny kontener (adres następnika). Dzięki temu wystarczy pamiętać wskaźnik do pierwszego elementu listy, aby pamiętać całą listę. Listy mogą zawierać jeden lub dwa wskaźniki, zgodnie z tym wyróżniamy **listy jednokierunkowe** i **listy dwukierunkowe**, które pozwalają na poruszanie się albo w jednym albo w obydwu kierunkach, materiał dotyczący list dwukierunkowych zostanie przerobiony na następnych zajęciach.

**Lista jednokierunkowa - schemat poglądowy**:

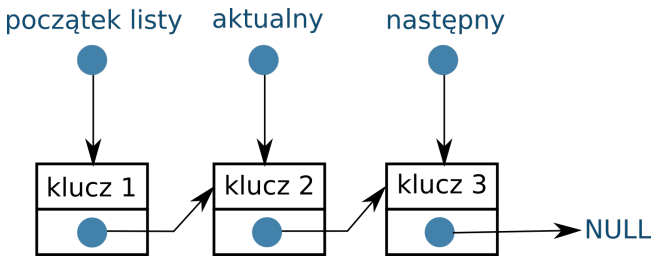


Figure 1: Schemat listy jednokierunkowej.

**Deklaracja listy jednokierunkowej**, w poniższym przykładzie zdefiniowano listę jednokierunkową, która zawiera następujące składowe: klucz identyfikacyjny i wskaźnik na następny węzeł (adres do następnika):

```
1 //definicja wezla na liscie: klucz i wskaznik na nastepnik
2 struct lista
3 {
4     int klucz;
5     struct lista *next; //pole wskaznikowe, adres do nastepnego wezla
6 };
```

Pierwszy i ostatni element listy nazywamy końcami listy. Aby mieć dostęp do całej listy wymagane jest utworzenie wskaźnika na początek listy na wartość NULL (ponieważ lista na starcie jest pusta), przykład deklaracji poniżej:

```
1 struct lista
2 {
3     int klucz;
4     struct lista *next; //pole wskaznikowe, adres do nastepnego wezla
5 };
6
7 lista *head = NULL; //wskaznik na poczatek listy
```

**Dodawanie nowego elementu**, jest podstawową operacją na liście. Elementy mogą być dodawane na początek listy, za wskazanym elementem lub na koniec listy. Dodając pierwszy element do pustej listy należy zapamiętać jego wskaźnik, by później mieć dostęp do całej struktury. Aby dodać nowy element w dowolnym miejscu zawsze

trzeba pamiętać aby ustawić wskaźnik aktualnego węzła na dodawany element, a wskaźnik tego elementu na następnik (aby zachować ciągłość struktury). Zwróć uwagę na przedstawione schematy dodawania elementu na początek, za wskazanym elementem i na koniec listy.

**Dodaj nowy element na początek listy**:

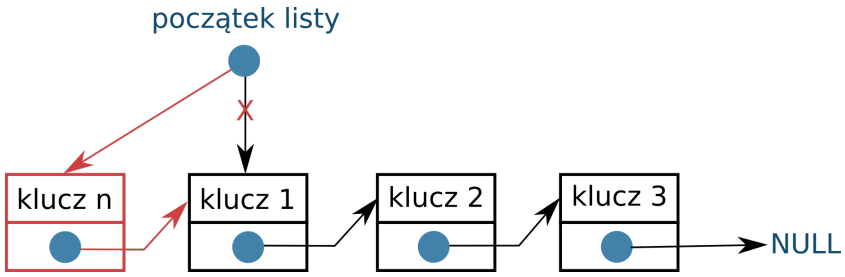


Figure 2: Dodawanie nowego elementu na początek listy.

**Dodaj nowy element za wskazanym elementem listy**:

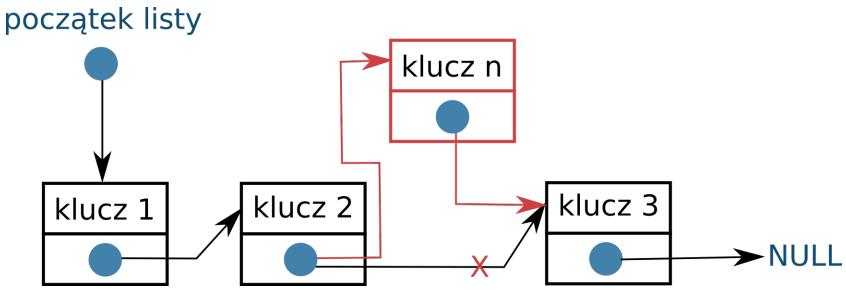


Figure 3: Dodawanie nowego elementu za wskazanym elementem listy.

**Dodaj nowy element na koniec listy**:

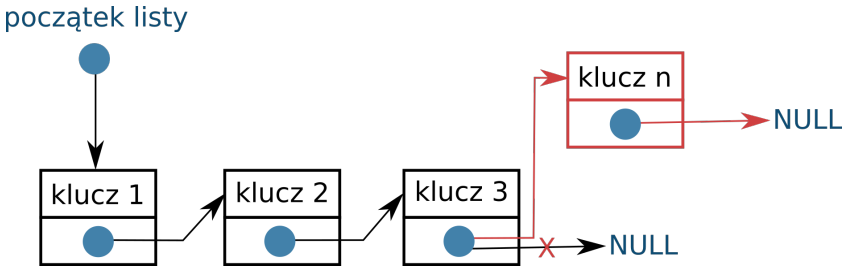


Figure 4: Dodawanie nowego elementu na koniec listy.

Poniżej przykładowa funkcja, która dodaje elementy na początek listy:

```
1  void dodajNaPoczatek(int liczba) //dodaj na poczatek
2  {
3      lista *element = new lista; //tworzymy nowy element struktury lista
4      //lista *element = (lista*)malloc(sizeof(lista));
5      element->klucz = liczba; //wskaznik na strukture wskazuje na pole klucz
6                               //polu klucz przypiszemy wartosc przekazana w argumencie
7      element->next = head; //element musi miec nastepnik
8      head = element; //nowy poczatek listy
9  }
```

Funkcja wyświetlająca:

```
1  void wyswietl(lista *head)
2  {
3      lista *element = head; //wskaznik do iterowania, wskazuje na poczatek listy
4      while (element != NULL) //wykonuj dzialania do konca listy
5      {                       //przerwij gdy element bedzie wskazywac na NULL
6          printf("%d ", element->klucz);
7          element = element->next;
8      }
9  }
```

**Usuwanie elementu**, jest kolejną podstawową operacją na listach. Podobnie jak przy dodawaniu elementu, usuwanie też można wykonać na kilka sposobów, usuwać możemy z początku listy, z końca listy, po wskazanej wartości klucza, czy po konkretnym indeksie (zależne od zdefiniowania zadania). Niezależnie od wyboru opcji usuwania należy odpowiednio ustawić wskaźniki, tak aby zachować ciągłość struktury. Zwróć uwagę na różnice przedstawione na poniższych schematach:

**Usuń element z początku listy**:

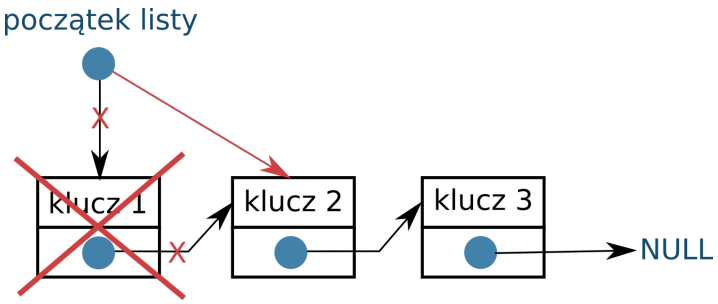


Figure 5: Usuwanie elementu z początku listy.

**Usuń element wskazany z listy**:

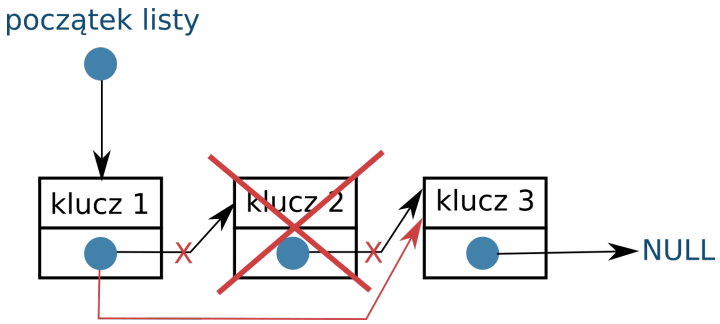


Figure 6: Usuwanie elementu wskazanego z listy.

**Usuń element z końca listy**:

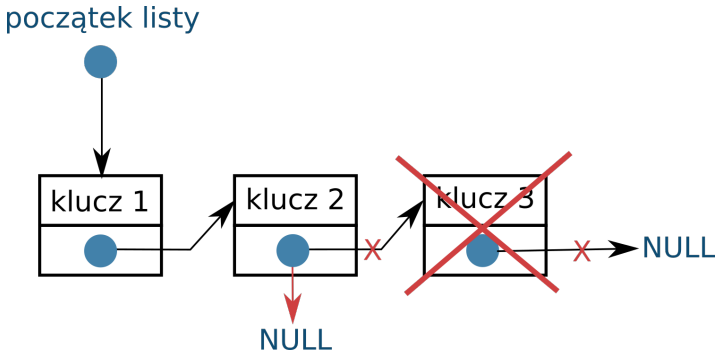


Figure 7: Usuwanie elementu z końca listy.

Poniżej przykładowa funkcja, która usuwa elementy z początku listy:

```
1  void usunZPoczatek() //usun z poczatku
2  {
3      lista *element = head;
4      if(element != NULL)
5      {
6          head = element->next; //nowy poczatek
7          delete element; //usun z pamieci C++ // free(element); //w C
8      }
9  }
```

Aby zapoznać się z działaniem listy jednokierunkowej zrealizuj poniższe zadania:

**Zad 1.** Wykorzystaj kod umieszczony w materiałach na stronie, przetestuj funkcje dodawania elementu na początek listy, usuwania elementu z początku listy. Następnie do programu dodaj funkcję dodawania elementu na koniec listy oraz funkcje usuwania elementu z końca listy i usuwania elementu o wskazanej wartości klucza.

## 2 Zadania

Zad 1. Zaprojektuj listę jednokierunkową służącą do przechowywania liczb całkowitych. Stwórz program umożliwiający obsługę takiej listy: możliwość dodawania, usuwania i modyfikowania elementów, wyszukiwania elementów listy, wyświetlania i (opcjonalnie) sortowania całej listy. Elementy powinny być dodawane na koniec listy.

Zad 2. Zaprojektuj listę jednokierunkową służącą do przechowywania rekordów (np. danych pracowników dowolnej firmy: id, imię, nazwisko, stanowisko). Stwórz program umożliwiający obsługę takiej listy: możliwość dodawania, usuwania, modyfikowania elementów, wyszukiwania, wyświetlania i (opcjonalnie) sortowania całej listy. Elementy powinny być dodawane na koniec listy.