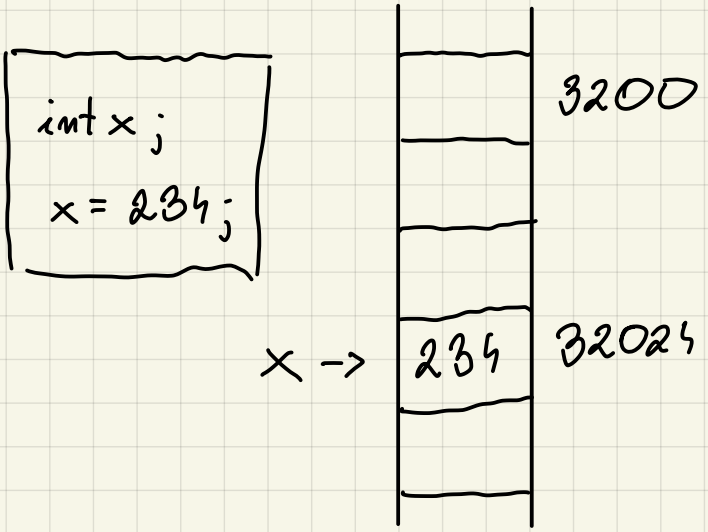
Zmienna i jej aspekty

Zmienna - miejsce w pamięci przechowujące wartość.

Aspekty zmiennej:
- nazwa
- adres (lokacja)
- wartość (układ bitów)
- typ zmiennej
- rozmiar



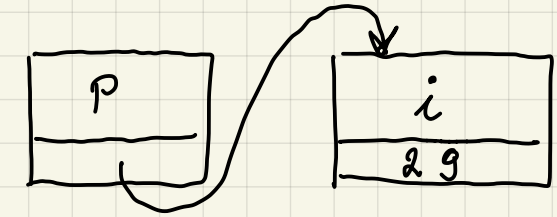
Funkcje transformujące: (język C)

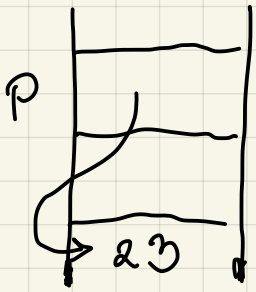
- zmienna -> adres : &x
- adres -> zmienna : *a

Zmienna wskaźnikowa

zmienna wskaźnikowa - zmienna przechowująca adres innej zmiennej

zmienna wskazywana - zmienna, na którą wskazuje zmienna wskaźnikowa



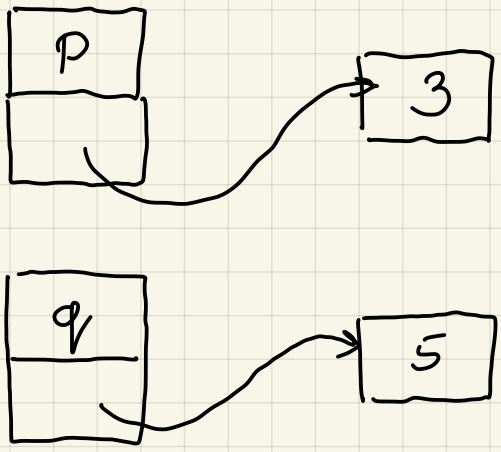


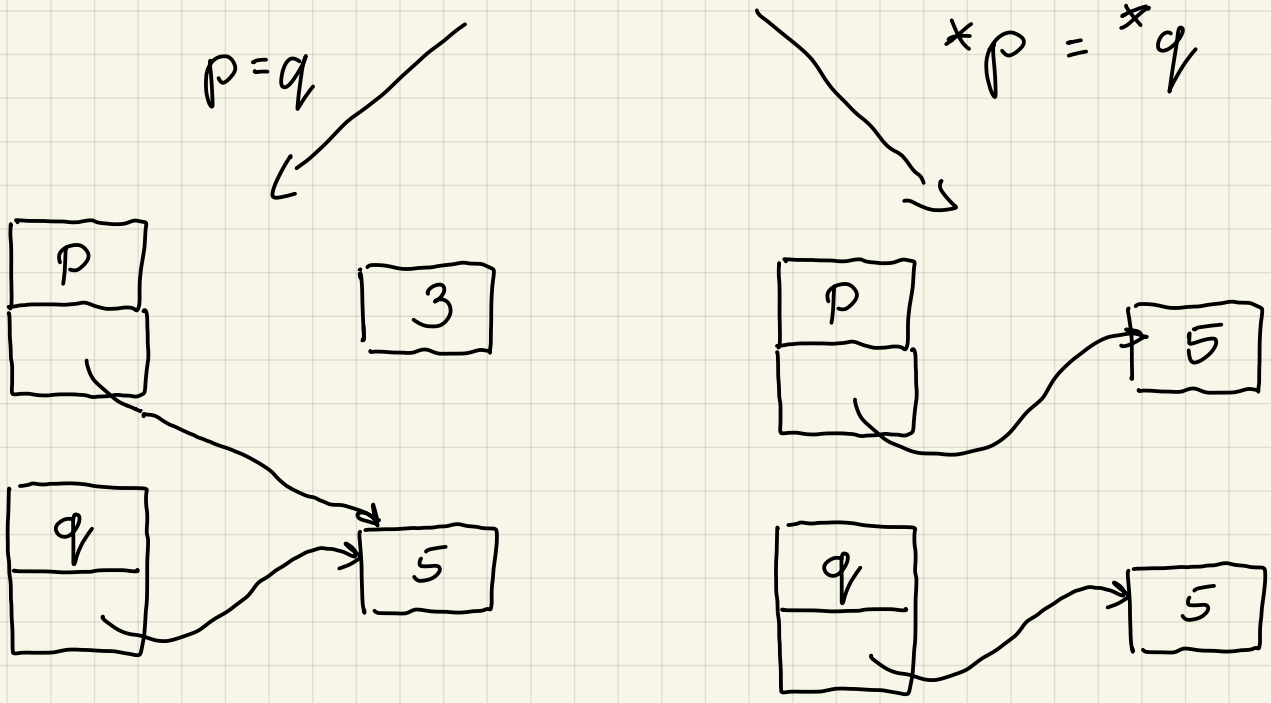
deklaracja zm. wskaźnikowej:

```
⋮
int i = 23;
int *p;

p = &i;
*p = 29;
```

```
{
    int *p;
    int *q;
    :
    *p = 3;
    *q = 5;
}
```





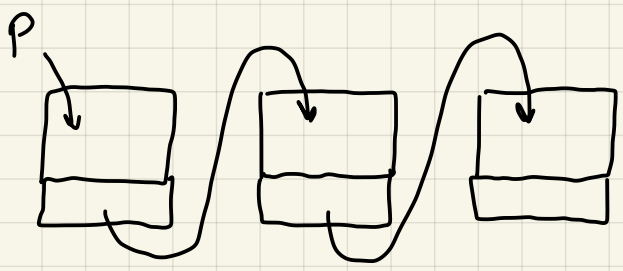
Operacje na zmiennych wskaźnikowych:

- deklarowanie: typ *p;
- alokacja zmiennej: p = new typ;
- zwalnianie pamięci: delete p;
- przypisanie: =
- operacje logiczne == !=
- wartość „pusta" NULL
- wypisanie wartości cout << p;

Zastosowanie typu wskaźnikowego

Nieregularne struktury danych:

- stos, kolejka, talia, lista
- struktura drzewiasta
- struktura grafowa

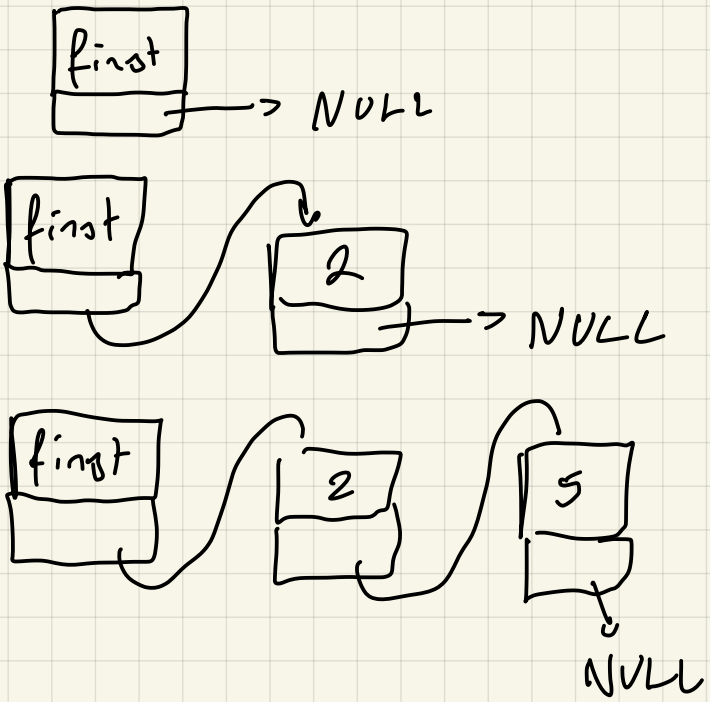


lista odsyłaczowa

Tworzenie łańcucha odsyłaczowego (listy)

```
struct node {
  int val;
  node *next;
};
  node *first;
  node *p;
  int s;
  first = NULL
  for (int i=1; i<n; i++){
     cin >> s;
     p = new node;
     p->next = first;
     p-> val = s;
     first = p;
  }
```

Etapy tworzenia listy:



Realizacja wskaźników w Pythonie

Programowanie obiektowe:

- obiekt - połączenie danych i operacji na nich wykonywanych, unikatowy egzemplarz danych zdefiniowanych w jego klasie
- metoda - funkcja określona w definicji klasy
- klasa - szablon, projekt, prototyp obiektu
- dziedziczenie - przekazywanie charakterystyki klasy do innych klas

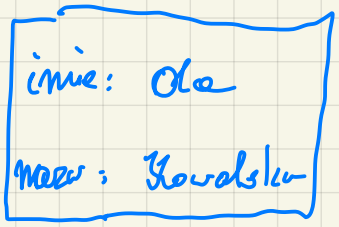
```
class osoba:
    pass

os1 = osoba()
os1.imie = "Ala"
os1.nazw = "Nowak"
```

```
class user:
    def __init__(self, imie, nazw):
        self.imie = imie
        self.nazw = nazw

u1 = user("Ola", "Kowalska")
```

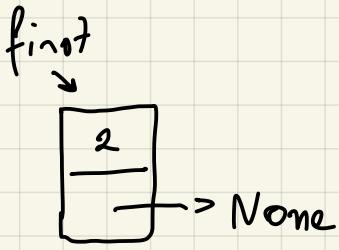
u1 ↑ wskaźnik na obiekt



Linked lists w Pythonie

```
class Node:
    def __init(self):
        self.val = None
        self.next = None
```

```
first = None
for i in range(4):
    s = int(input('>'))
    p = Node()
    p.val = s
    p.next = first
    first = p
```

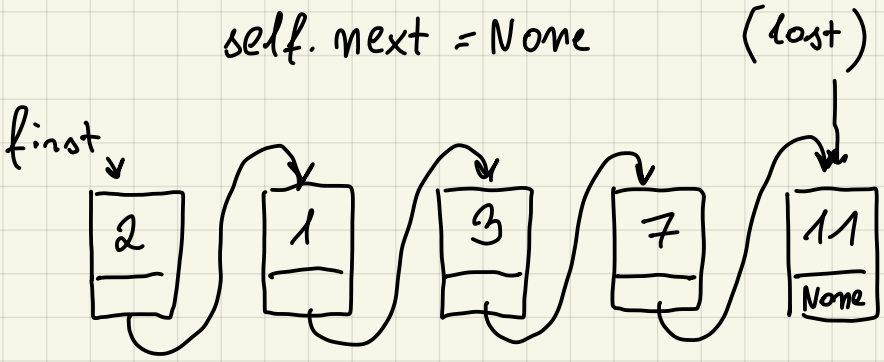



```
def wypisz1(p):
    while p != None:
        print(p.val)
        p = p.next
```

- lista jednokierunkowa

```
class Node:
    def __init__(self, val)
        self.val = val
        self.next = None
```

C++	Python
p->val	p.val
p= new Node	p=Node()
p= p->next	p= p.next
delete p	—



Elementarne operacje:

- dołączenie nowego elementu na początek:

```
p=Node()
p.val = 1
p.next = first
first = p
```

- dołączenie nowego elementu na koniec

```
p=Node()
p.val = 13
r = first
while r.next != None:
    r = r.next
r.next = p
```

- wyszukiwanie

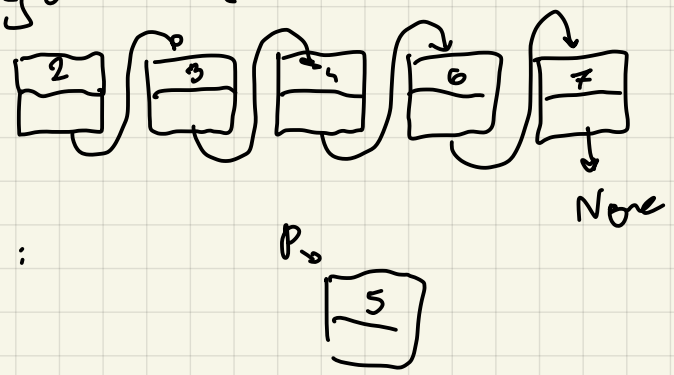
```
# lista nieposortowana
def find(first, key):
    p= first
    while p!=None:
        if p.val == key:
            return p
        p= p.next
```

```
# lista posortowana
def find(first, key):
    p= first
    while p!=None and p.val < key:
        p=p.next
    if p==None: return None
    if p.val == key: return p
    return None
```

- wstawianie wartości (przed lub za daną elementu)

- wstawianie elementu do posortowanego łańcucha

```
def insert(head, key):
    new_node = Node(key)
    if head == None or head.val > key:
        new_node.next = head
        return new_node
    else:
        first = head
        while head.net != None and head.next.val < key:
            head = head.next
        new_node.next = head.next
        head.next = new_node
        return first
```



- Lista dwukierunkowa

```
class Node:
    def __init(self):
        self.val = None
        self.next = None
        self.prev = None
```