

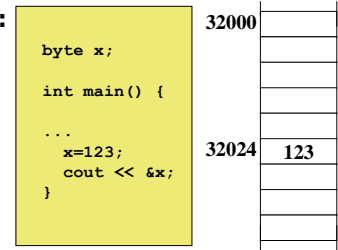
Wykład 8

- Zmienna i jej aspekty
- Zmienna wskaźnikowa
- Przydział pamięci dla zmiennych
- Działania na zmiennych wskaźnikowych
- Zastosowanie typu wskaźnikowego
- Przykłady

Zmienna

Aspekty zmiennej:




- nazwa
- adres (lokacja)
- wartość
- typ
- rozmiar



1

2

Czas istnienia nazwy

Program w C/C++  kompilacja  Program wykonywalny
nazwa zmiennej  adres w pamięci

```
procedure main()
  i:=5; j:=7; k:=11
  nazwa:=read()
  m:=variable(nazwa)
  write(m)
  variable(nazwa):=3
  write(j)
end
```

Język Icon

Zmienna wskaźnikowa

Zmienna wskaźnikowa - zmienna, która przechowuje adres innej zmiennej.

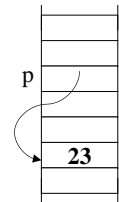
Zmienna wskazywana - zmienna, na którą wskazuje zmienna wskaźnikowa.



Deklaracja zmiennej wskaźnikowej:

```
int i=23;
int *p;

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

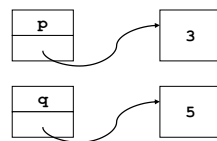


3

4

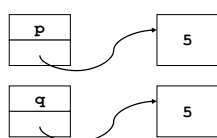
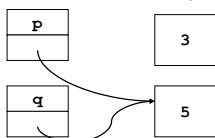
Zmienna wskaźnikowa i wskazywana

```
int *p;
int *q;
...
//alokacja zmiennych
...
*p = 3;
*q = 5;
...
```



p = q

*p = *q

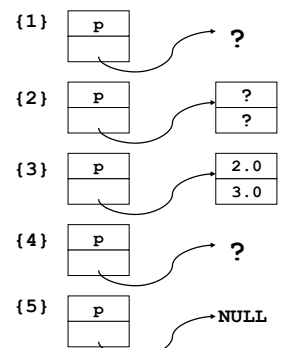


Alokacje i zwalnianie pamięci

```
struct punkt {
  double x,y;
};

punkt *p;

p = new punkt;
...
p->x = 2.0;
p->y = 3.0;
...
delete p;
p = NULL;
```



5

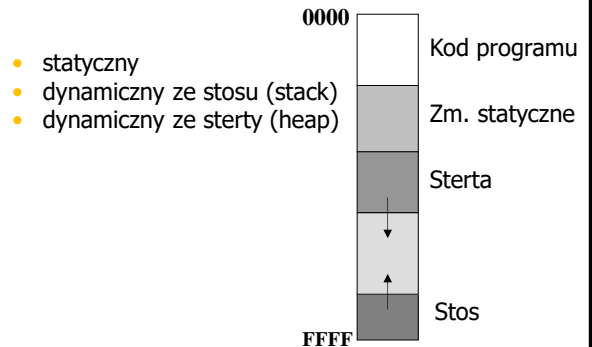
6

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;`

7

Przydział pamięci



8

Zastosowania typu wskaźnikowego

Dynamiczne struktury danych:

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

9

Tworzenie łańcucha odsyłaczowego (listy)

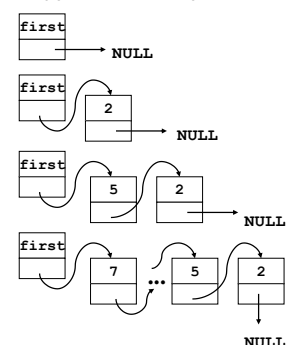
```

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:



10

Wypisanie wartości listy

Iteracyjnie

```

void wypisz1(node *p)
{
    while (p!=NULL)
    {
        cout << p->val;
        p=p->next;
    }
}
    
```

Rekurencyjnie

```

void wypisz2(node *p)
{
    if (p!=NULL)
    {
        cout << p->val;
        wypisz2(p->next);
    }
}
    
```

11

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

12

Programowanie obiektowe

```
class osoba:
    pass
# end

os1 = osoba()
os1.imie = 'Ala'
os1.nazw = 'Nowak'

class user:
    def __init__(self, imie, nazw):
        self.imie = imie
        self.nazw = nazw
    # end
# end
u1 = user('Ola', 'Kowalska')
```

13

Lista

```
class Node:
    def __init__(self, val):
        self.val = None # przechowywana liczba całkowita
        self.next = None # odnośnik do następnego elementu

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

wypisz1(first)
```

14

Wypisanie wartości listy

Iteracyjnie

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

Rekurencyjnie

```
def wypisz2(p):
    if p!=None:
        print(p.val)
        wypisz2(p.next)
    end
end
```

15

Więcej

Linked Lists in Python: An Introduction
<https://realpython.com/linked-lists-python/>

16