Ochrona zmiennych przekazywanych do funkcji

- Jeśli do funkcji przekazywana jest prosta zmienna, na przykład typu `int`, programista może przekazać do funkcji wartość zmiennej lub jej wskaźnik. Powszechnie stosuje się zasadę, że przekazuje się wartość, chyba, że występuje konieczność zmiany tej wartości, wtedy bądź przekazywany jest adres, bądź korzysta się z mechanizmu referencji.
- W przypadku tablic nie ma wyboru: zawsze przekazywany jest adres. W rezultacie funkcja przetwarzająca tablicę działa na danych pierwotnych i może je swobodnie zmieniać. Czasem jest to pożądane, ale wiele funkcji tylko czyta tablice i nie modyfikuje ich zawartości. Jednakże, jeśli programista popełni błąd, może zniszczyć dane zapisane w tablicy.
- Przykład:

```
#include <iostream.h>

// Funkcja poprawna
int suma(int t[], int n) {
  int suma=0;
  for (int i=0; i< n; i++)
    suma += t[i];
  return suma;
}

int main() {
  int a[10]={1,3,2,5,4,8,9,2,4,2};
  cout << "suma = " << suma(a,10)
       << endl;
  return 0;
}
```

```
#include <iostream.h>

// Funkcja błędna
int suma(int *t, int n) {
  int suma=0;
  for (int i=0; i< n; i++)
    suma += t[i]++;
  return suma;
}

int main() {
  int a[10]={1,3,2,5,4,8,9,2,4,2};
  cout << "suma =" << suma(a,10)
       << endl;
  return 0;
}
```

Jeśli wiemy, że wartości elementów tablicy nie mogą być w funkcji zmieniane, możemy zadeklarować argument formalny funkcji z użyciem słowa kluczowego `const`. Czyli nagłówek funkcji może mieć postać:

```
int suma(const int t[], int n)
```

lub

```
int suma(const int *t, int n)
```

Oznacza to, że *w funkcji* tablica t ma być tak traktowana, jakby zawierała stałe dane. Próba kompilacji wersji 2 funkcji z tym nagłówkiem zakończy się komunikatem:

```
Cannot modify a const object.
```

Zastosowanie wskaźników - dynamiczne przydzielanie pamięci

- Pamięć dla zmiennych może być przydzielana:
  - *statycznie* - przydziela ją kompilator, przetwarzając kod źródłowy programu
  - *dynamicznie* - przydzielają ją odpowiednie funkcje biblioteczne wywoływane podczas wykonywania programu
- Przydział statyczny
  - program jest wydajniejszy, ponieważ pamięć jest przydzielona zanim program zacznie działać
  - program jest mniej elastyczny ,ponieważ przed wykonaniem musimy znać wielkość wymaganej pamięci..
- Przydział dynamiczny
  - program jest bardziej elastyczny i mniej wydajny
- Obiekty statyczne (zmienne) mają nazwy i odwołujemy się do nich bezpośrednio.
- Obiekty dynamiczne nie mają nazwy. są umieszczane w obszarze pamięci nazywanym *stertą* (ang. *heap*), a dostęp do nich uzyskuje się za pomocą wskaźnika do zajmowanego przez nie bloku pamięci.

Przydzielanie pamięci w języku C++

- Do przydzielania pamięci służy operator `new`.
- Do zwalniania przydzielonej pamięci służy operator `delete`.
- Przykład 1: proste zmienne

```
int *wi=new int; // przydzielenie pamięci dla jednej zmiennej typu int
delete wi; // zwolnienie pamięci, na którą wskazywał wi
```

- Domyślnie obszar przydzielonej pamięci nie jest zainicjowany. Jeśli chcemy przypisać wartość zmiennej podczas inicjalizacji musimy to zrobić jawnie:

```
int *wi=new int(0); // przydzielenie pamięci dla jednej zmiennej typu
int
                // przypisanie tej zmiennej wartości początkowej 0
```

- Przykład 2: tablice

```
int *wti=new int [100]; // przydzielenie pamięci dla tablicy typu int
delete[] wti; // zwolnienie pamięci, na którą wskazywał wi

int (*wti2)=new int [4][100]; // przydzielenie pamięci dla tablicy typu
int
delete[] wti2; // zwolnienie pamięci, na którą wskazywał wi
```

- Rozmiar tablicy, która ma być umieszczona w pamięci przydzielanej dynamicznie, może być wartością obliczoną podczas wykonywania programu.

```
int *wti3=new int[ile];
int *wti4=new int[pobierzWymiar()];
int (*wti5)[100]=new int[pobierzWymiar()][100];
```

- Jeżeli chcemy zainicjować elementy tablicy w pamięci przydzielanej dynamicznie, musimy przypisać wartości poszczególnym elementom.

```
int *wtab = new int[100';
if (wtab != 0)
  for (int i=0; i<100; ++i)
    wtab[i]=100;
else
  cout << "Tablicy nie udało się utworzyć"<<endl;
```

Tablice wskaźników

- Tablica wskaźników to jednowymiarowa tablica, której elementami są wskaźniki.
- Przykład deklaracji tablicy 10 wskaźników do typu `int`:

```
int *x[10];
```

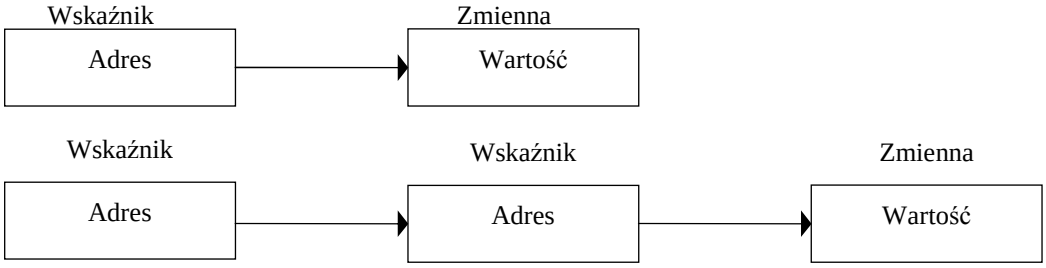
- Przypisanie wartości elementowi tablicy:

```
x[0]=&a;
```

- Tablice wskaźników są stosowane na przykład przy przetwarzaniu tekstów. Patrz: Napisy.

Wskaźnik do zmiennej wskaźnikowej

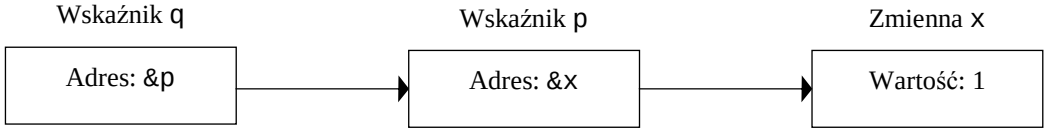
- Wartością zwykłego wskaźnika jest adres miejsca pamięci, w którym znajduje się wartość zmiennej.
- Wartością wskaźnika do wskaźnika jest adres miejsca pamięci, w którym znajduje się wskaźnik wskazujący miejsce pamięci z wartością zmiennej.



- Deklaracja wskaźnika do wskaźnika:

```
int **wsk;
```

- Dostęp do wartości zmiennej wskazywanej przez wskaźnik do wskaźnika:




```
#include <iostream.h>
int main()
{
  int x, *p, **q;
  x=1;   // przypisanie wartości zmiennej
  p=&x;  // przypisanie wartości wskaźnikowi zwykłemu
  q=&p;  // przypisanie wartości wskaźnikowi do wskaźnika
  cout << **q << endl; // wyświetlenie wartości zmiennej
  return 0;
}
```

Referencje

- *Referencja* (odniesienie, ang. *reference*) to inna nazwa obiektu (zmiennej).
- Technicznie można powiedzieć, że referencja jest pewnym rodzajem wskaźnika używanego bez specjalnej składni wskaźnika.
- Referencję deklarujemy za pomocą operatora & umieszczonego między nazwą typu i nazwą referencji:

```
int a=123;   // obiekt typu int
int *wsk=&a; // wskaźnik do obiektu
int &ra=a;   // referencja do obiektu
```

- Referencja MUSI być zainicjalizowana:

```
int a=1;
int &r1=a;  // OK.
int &r2;    // BŁĄD! Brak inicjatora
int &r3=&a  // BŁĄD! Nie można inicjować adresem
```

- Raz zdefiniowana referencja nie może być później zmieniana tak, aby odnosiła się do innego obiektu.
- Wszystkie działania wykonywane na obiekcie referencyjnym działają na tym obiekcie, do którego odnosi się dany obiekt referencyjny.

```
int a=0,b=1,c=1,*wa=&a; // deklaracja wskaźnika
int x=0,y=1,z=1,&rx=x;   // deklaracja referencji

wa=&b;  // zmiana wskaźnika
rx=y;   // przypisanie zmiennej x wartości zmiennej y, czyli x=y
```

- W praktyce rzadko stosuje się samodzielne obiekty referencyjne. Najczęściej używa się referencji jako argumentów formalnych funkcji.

Referencje jako argumenty formalne funkcji

- Deklaracja referencyjnego argumentu formalnego unieważnia mechanizm domyślnego przekazywania argumentów przez wartość. Funkcja wie, gdzie dany argument znajduje się w pamięci i może dzięki temu zmienić jego wartość.

- *Przykład 1*. Referencja może zastąpić wskaźniki, jeżeli chcemy aby zmiana wartości była widoczna w funkcji wywołującej.

```
# include <iostream.h>

// funkcja korzysta ze wskaźników
void swap1(int *x, int *y) {
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}

int main() {
  int a=6,b=3;
  swap1(&a,&b);
  cout<<"Zamienione liczby 6,3\t"
      <<a <<'\t' <<b <<endl;
return 0;
}
```

```
# include <iostream.h>

// funkcja korzysta z referencji
void swap2(int &x, int &y) {
   int temp
   temp=x;
   x=y;
   y=temp;
}

main() {
  int c=4,d=8;
  swap2(c,d);
  cout <<"Zamienione liczby 4,8\t"
       <<c <<'\t' <<d <<endl;
  return 0;
}
```

- *Przykład 2*. Wykorzystanie referencji do przekazywania dodatkowego wyniku do funkcji wywołującej. Funkcja `szukaj()` przeszukuje tablicę liczb całkowitych `t` o rozmiarze `n` i poszukuje w niej liczby `x`. Liczby w tablicy mogą się powtarzać. Jeśli liczba zostanie znaleziona, funkcja zwraca indeks wskazujący pierwsze wystąpienie liczby `x`. Jeśli liczba nie zostanie znaleziona przekazywana jest liczba przeszukanych elementów tablicy. Dodatkowo za pośrednictwem parametru referencyjnego występuje przekazywana jest informacja o liczbie wystąpień szukanej wartości.

```
int szukaj(int *t, int n, int x, int &wystepuje)
{
  int indeks=n;
  wystepuje=0;
  for (int i=0; i<n; ++i)
    if (*(t+i) == x)
    {
      if (indeks==n)
        indeks=i;
      ++wystepuje;
     }
   return indeks;
}
```

Struktury

- Struktura to zbiór zmiennych tego samego lub różnego typu występujących pod wspólną nazwą. Umożliwia ona grupowanie informacji ze sobą powiązanych.
- Składnia deklaracji struktury:

```
struct etykieta_struktury
{
   typ_1 nazwa_składowej_1;
   typ_2 nazwa_składowej_2;
   ...
   typ_N nazwa_składowej_N;
};
```

gdzie:
- etykieta identyfikuje dany rodzaj struktury i może być użyta jako skrót tej części, która występuje w nawiasach klamrowych.
- dane struktury nazywane są *składowymi*, *polami* lub *elementami* (ang. *members*, *fields*, *elements*).

- Nazwy składowych danej struktury muszą być różne.
- Każda składowa struktury jest określonego typu (np. `int`, `float`, `char`),
- Deklaracja struktury kończy się średnikiem.
- Przykład 1a: struktura opisująca punkt oraz dwie zmienne typu strukturalnego:

```
struct Punkt{
  int x;
  int y;
} p1,p2;
```

etykieta struktury

składowe struktury

lista zmiennych typu Punkt

- Przykład 1b: deklaracja struktury może być oddzielona od deklaracji zmiennych:

```
struct Punkt{
  int x;
  int y;
};

Punkt p1,p2;
```

Komentarz:
- Deklarując zmienne typu struktury, w języku C++ można pominąć słowo kluczowe `struct`.
- W języku C trzeba podać słowo `struct`, czyli deklaracja zmiennych ma postać:
  `struct Punkt p1,p2;`
- Jeżeli chcemy posługiwać się w C składnią podobną do tej w języku C++, możemy posłużyć się instrukcją `typedef`:

```
typedef struct {
  int x;
  int y;
} Punkt;
Punkt p1,p2;
```

- Przykład 2: struktura opisująca pracownika:

```
struct Osoba {
  char imie_nazwisko[30];
  int wiek;
  float placa;
};
Osoba ksiegowy, sekretarka;
Osoba pracownicy[10]; /* deklaracja tablicy z elementami typu osoba
  */
```

Inicjalizowanie struktur

- Struktury inicjalizuje się podobnie do inicjalizacji tablic. Dopisuje się na końcu definicji listę wartości początkowych składowych struktury.

```
struct Punkt
{  int x;
   int y;
};
Punkt p1 = {1,1}; // Uwaga: nawiasy klamrowe!

struct Adres {
  char imie_nazwisko[30];
  int wiek;
  float placa;
};
Adres pracownik = {"Adam Nowak", 25, 1200};

struct Ulamek {
  long Licznik,Mianownik;
} A,B,Zerowy={0,0}; // ułamki A i B nie mają jeszcze wartości
Ulamek C,D={1,4},E={6,8}; // zainicjowanie tylko ułamków D i E
```

Dostęp do składowych struktury

- Aby odwołać się do składowej struktury, trzeba użyć *operatora wyboru składowej* oznaczanego *kropką*.
- Składnia:

  nazwa_zmiennej_strukturowej.nazwa_składowej

- Przykład:

```
struct punkt
{  int x;
   int y;
};
punkt p1,p2;

// przypisywanie wartości
p1.x=1;
p1.y=1;
// wyświetlanie
cout << "Punkt p1(" << p1.x << ',' << p1.y << ')' << endl;
// wczytywanie
cin >> p2.x >> p2.y;
// użycie w wyrażeniu
odl=sqrt((double)p1.x*p1.x + (double)p2.y*p2.y);

struct osoba{
  char imie_nazwisko[30];
  int wiek;
  float placa;
};
osoba pracownik;
cout << "Podaj imie i nazwisko: ";
cin.getline(pracownik. imie_nazwisko,30);
cout<< pracownik. imie_nazwisko << endl;
```

Wzajemne przypisania struktur

- Dane zawarte w jednej strukturze można przypisać innej strukturze tego samego typu za pomocą przypisania.
- Przykład:

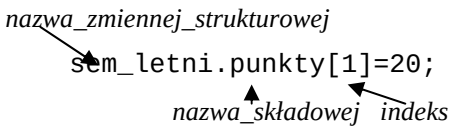
```
#include <iostream.h>
struct punkt {
  int x;
  int y;
};
int main() {
  punkt p1, p2;
  p1.x=10;
  p1.y=20;
  cout <<"Wspolrzedne punktu p1: " << p1.x << ',' << p1.y << endl;
  p2=p1; // przypisanie jednej struktury innej
  cout <<"Wspolrzedne punktu p2: " << p2.x << ',' << p2.y << endl;
  return 0;
}
```

Struktury z tablicami

- Elementem struktury może być tablica.

```
struct oceny {
  long nr_indeksu;
  int ile_testow;
  int punkty[10];
  int ocena[10];
};
struct oceny sem_letni;
```

- Dostęp do elementu tablicy:



- Przykład:

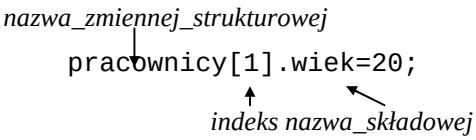
```
#include <iostream.h>
struct Oceny {
  unsigned int indeks;
  int ile_testow;
  int punkty[10];
  int ocena[10];
};

int main(){
  Oceny sem_letni={1256,3,{28,15,10},{5,3,2}};
  cout << "Student nr indeksu: " << sem_letni.indeks
       << "  Liczba testow:" << sem_letni.ile_testow << endl;
  for (int i=0;i<sem_letni.ile_testow;i++)
    cout << "    Test nr " << (i+1) << " Punkty=" <<
sem_letni.punkty[i]
         << " Ocena=" << sem_letni.ocena[i] << endl;
  return 0;
}
```

Tablice struktur

- Zamiast wielu zmiennych można używać tablicy struktur.

```
struct Osoba {
  char imie_nazwisko[30];
  int wiek;
  float placa;
};
Osoba pracownicy[10];
```

- Dostęp do elementu tablicy:

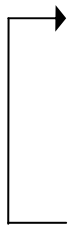


- Przykład:

```
#include <iostream.h>
#include <stdio.h>
  struct Osoba {
    char imie_nazw[30];
    double placa;
  };
int main()
{
  int i;
  int znak;
  Osoba pracownicy[10];
  for (i=0;i<10;i++)
  {
    cout << "Wpisz nazwisko: ";
    cin.getline(pracownicy[i].imie_nazw,sizeof(pracownicy[i].imie_nazw);
    cout << "Wpisz place: ";
    cin >> pracownicy[i].placa;
    while ( (znak=cin.get()) != '\n');
  }
  cout << "Nazwisko" <<'\t' << "Placa" << endl;
  for (i=0;i<10;i++)
    cout << pracownicy[i].imie_nazw << '\t' <<
      pracownicy[i].placa << endl;
  return 0;
}
```

Struktury zawierające struktury

- Elementem składowym struktury może być inna struktura.



```
struct Data {
  int dzien;
  int miesiac;
  int rok;
};
struct Dzien {
  char dzien_tygodnia[15];
  Data data;
};

Dzien biezacy={"Poniedzialek",{31,5,2001}};
cout << "Biezaca data: " << biezacy.data.dzien <<'.'
                         << biezacy.data.miesiac << '.'
                         << biezacy.data.rok << endl;

Dzien nastepny;

strcpy(nastepny.dzien_tygodnia,"Wtorek");
nastepny.data.dzien=1;
nastepny.data.miesiac=6;
nastepny.data.rok=2001;
cout << "Nastepna data: " << nastepny.data.dzien <<'.'
                          << nastepny.data.miesiac << '.'
                          << nastepny.data.rok << endl;
```

Wskaźniki do struktur

- Deklarowanie wskaźników do struktur

```
nazwa_struktury *nazwa_zmiennej;
```

- Przykład:

```
struct Data {
  int dzien;
  int miesiac;
  int rok;
};
Data data_biezaca, tydzien[7];
Data *wsk_data;
```

- Korzystanie ze wskaźników do struktur

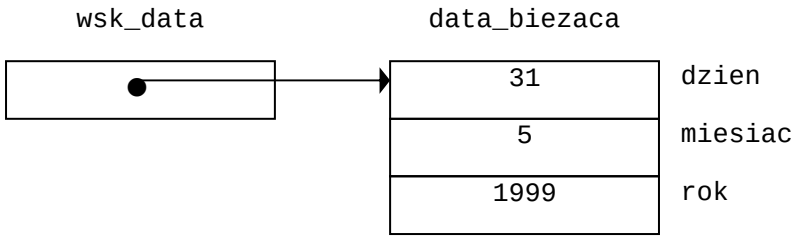
```
// Adres zmiennej strukturowej uzyskiwany jest za pomocą operatora &
  wsk_data=&dzien_biezacy;
  wsk_data=&tydzien[0];
// Przydzielenie pamięci na zmienną strukturową
  wsk_data=new Data; // C++
  wsk_data= (struct Data *) malloc(sizeof(struct Data); // C
// Zwolnienie pamięci
  delete wsk_data;
  free(wsk_data);
```

- Dostęp do wartości pól struktury uzyskiwany jest za pomocą operatora `->` .
  - Zamiast pisać:

```
(*wsk_data).dzien=31;
```

    można:

```
wsk_data->dzien=31;
wsk_data->miesiac=5;
wsk_data->rok=1999;
```



- Przykład : przekazywanie struktury za pomocą wskaźnika

```
#include <iostream.h>
struct Pomiary {
  char nazwa;
  int a,b;
};
void drukuj(Pomiary *proba);
int main() {
  Pomiary proba;
  proba.nazwa='A';
  proba.a=100;
  proba.b=20;
  drukuj(&proba);
  return 0;
}
void drukuj(Pomiary *proba) {
  cout << "Nazwa: "<< proba->nazwa << '\t'
       <<" Wyniki: " << proba->a << '\t'
       << proba->b << endl;
  }
```

- Przykład: dynamiczne przydzielanie pamięci

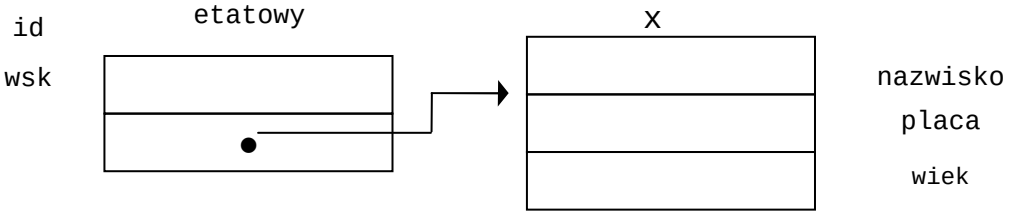
```
#include <iostream.h>
struct Osoba {
  char nazwisko[30];
  int placa;
};
void czytaj(Osoba *wsk_osoba);
void drukuj(Osoba *wsk_osoba, int nr);
int main()
{
  int znak;
  int ile;
  Osoba *pracownicy;
  cout << "Ilu pracownikow? ";
  cin >> ile;
  //  wyczysc;
  while ( (znak=cin.get()) != '\n');
  pracownicy=new Osoba[ile];
  for (int i=0;i<ile;i++)
    czytaj(&pracownicy[i]);
  for (int i=0;i<ile;i++)
    drukuj(&pracownicy[i],i+1);
  delete [] Osoba;
  return 0;
}
void czytaj(Osoba *wsk_osoba)
{
    int znak;
    cout << "Wpisz nazwisko: ";
    cin.getline(wsk_osoba->nazwisko,30);
    cout << "Wpisz place: ";
    cin >> wsk_osoba->placa;
    while ( (znak=cin.get()) != '\n');
}
void drukuj(Osoba *wsk_osoba, int numer)
{
cout << numer <<'.'
     << wsk_osoba->nazwisko << ' ' << wsk_osoba->placa <<endl;
}
```

Struktury zawierające wskaźniki

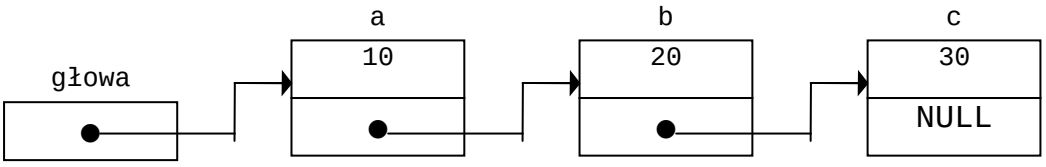
- Składowymi struktury mogą być wskaźniki.

- Przykład:

```
struct Osoba {
  char nazwisko[30];
  int placa;
  int wiek;
};
struct Pracownik {
  int id;
  struct Osoba *dane;
};

Pracownik etatowy;
Osoba x;

etatowy.dane=&x;
etatowy.dane->wiek=20;
```



Przykład: lista jednokierunkowa



Do budowy listy używane są dwa typy elementów:

- element informacyjny - zawiera wskaźnik do początku listy
- element z danymi - zawiera dane oraz wskaźnik do następnego elementu listy.

```
struct Element {
  int wartosc;
  struct element *nastepny;
};
Element a,b,c;
Element *glowa;
```

```
glowa=&a;
a.nastepny=&b;
b.nastepny=&c;
c.nastepny=0;
```

- Przykład:

```
#include <iostream.h>

struct Element {
  int wartosc;
  struct Element *nastepny;
};

main() {
  Element *glowa=0;
  int ile;

  // utwórz listę
  cout << "Ile elementów będzie miała lista? ";
  cin >> ile;

  for (int i=0;i<ile;i++) {
    Element *nowy=new Element;
    nowy->nast=glowa;
    glowa=nowy;
    cout << "Wpisz wartosc elementu: ";
    cin >> glowa->wartosc;
  }
// wyświetl zawartość listy
  cout << endl;
  Element *biezacy=glowa;
  while (biezacy) {
    cout << "Element: " << biezacy->wartosc << endl;
    biezacy=biezacy->nastepny;
  }
// usuń listę
  biezacy=glowa;
  while(biezacy) {
    Element *usuwany=biezacy;
    biezacy=biezacy->nastepny;
    delete biezacy;
  }
  return 0;
}
```

- Pytanie: Gdzie będzie znajdował się element wstawiony jako pierwszy do listy - na początku czy na końcu listy? Głowa wskazuje początek listy.